EXPLANATION-BASED FEATURE CONSTRUCTION

BY

SHIAU HONG LIM

B.C.S., University of Malaya, 2000
M.C.S., University of Malaya, 2001

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2009

Urbana, Illinois

Doctoral Committee:

Professor Gerald DeJong, Chair
Professor Dan Roth
Professor David Forsyth
Assistant Professor Eyal Amir

Abstract

Incorporating additional information from our prior domain knowledge can be the key to solving difficult classification tasks, especially when the available training data is limited. The crucial stage of feature construction, often done manually, plays a significant role in allowing such information to be incorporated into a learner.

We propose algorithms for automated feature construction where available domain knowledge, even though imperfect and approximate, can be utilized by the learning system. Robustness is achieved by incorporating this prior knowledge in a task-specific manner, guided by the actual training examples. These goals are realized with Explanation-Based Learning (EBL).

The EBL paradigm provides the necessary bridge between domain knowledge and the training examples, which allows us to design solutions that are conceptually well-formed and *work for the right reason*. The ideas of well-formed concepts and "working for the right reason" are our guiding principles for supervised learning.

Using these underlying principles, we propose three algorithms for incorporating prior domain knowledge into discriminative learning with different levels of interaction between the feature construction process and the final classifier learning. The first approach involves automated construction of generative models for *phantom examples*, which can be used to enhance the training data for subsequent classifier learning. Both the second and the third approaches involve the construction of *semantic features*. Each semantic feature encapsulates a well-formed concept which, according to the domain knowledge, corresponds to a conceptual difference between classes of objects.

We illustrate and evaluate the proposed algorithms on the challenging problem of classifying offline handwritten Chinese characters, focusing on distinguishing difficult, mutually-similar pairs of characters. Empirical results show that our approaches can outperform the state-of-the-art algorithms.

To my parents

Acknowledgments

I am deeply grateful to my advisor, Gerald DeJong for his guidance and support throughout the years. His enthusiasm for research, especially on EBL, is unmatched and has become a constant source of motivation. His insights and inspirations often shed new light on alternative ways to address a problem and open up new research directions, eventually allowing me to complete my PhD.

I also wish to thank all the members of the EBL group that I have the privilege of working with: Li-Lun Wang, Geoffrey Levine, Arkady Epshteyn, Qiang Sun and Adam Laud. Each person has helped me in unique ways and has offered me invaluable comments on my research. It is a great pleasure sharing thoughts and working with them.

Finally, I thank my parents for their immeasurable love and support. They had great confidence in me and allowed me complete freedom to pursue my own passions. This work is dedicated to them.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Models, Features and Learning

Machine learning has been successfully applied in a large variety of domains, from natural language processing to search engines, from bioinformatics to computer vision and so on. On the other hand, we have yet to achieve human-level performance in many tasks, even seemingly simple ones from a human point of view, such as handwriting recognition.

The proven futility of bias-free learning [Mitchell, 1980, 1997], as well as the no-free-lunch theorems [Wolpert, 1996, 2001], constantly remind us that the successful application of machine learning must be attributed to carefully chosen inductive bias, which should naturally come from our prior knowledge about the application domains. The incorporation of prior knowledge into learning systems, however, remains a challenging and largely unsolved problem.

We believe that for many learning tasks, the very basic stage of model or feature construction plays the biggest role in determining its success. This is also where most of our prior knowledge is needed. Many, if not most, successful applications of machine learning can be attributed to carefully crafted models and features that fit particularly well to the target domains.

We pursue a direction where at least part of this crafting process can be automated. We focus specifically on classification problems, where the construction of good features is crucial. Indeed, for many classification problems, once the "right" feature set is constructed, the problem is essentially solved.

The challenge of automated, data-dependent feature construction is often in avoiding overfitting while entertaining an expressive space of possible solutions, especially when the available training data is limited. We seek a principled method that allows us to incorporate rich prior domain knowledge into this process.

1.2 Explanation-Based Learning

Consider the set of all solutions that fit the training data well. In an expressive solution space this set can be huge. However, in many cases, the number of solutions that "make sense" to a human expert cannot be too large. In other words, the set of solutions that "work for the right reason" is necessarily small.

Exactly which solution works for the right reason depends entirely on the expert's knowledge. We can think of encoding our prior domain knowledge into a *domain theory*, which implicitly defines the set of *reasonable* or *well-formed* solutions. Although the space of reasonable solutions can still be very large, the ones that actually work (i.e. fit well) on a particular training set may be very small.

Explanation-Based Learning (EBL) [DeJong and Mooney, 1986; Mitchell et al., 1986; DeJong, 2006] can be viewed as a learning paradigm where the training examples are used to expose the set of reasonable solutions from an expert-defined domain theory. Finding this set of reasonable solutions is called *explanation*. Any residual uncertainty can be resolved by standard statistical techniques. Given that the set of reasonable solutions is small, which is often an indication that the domain theory is adequate, a small training set may be sufficient.

1.3 Offline Handwritten Character Recognition

While handwriting recognition systems for *online* or machine-printed handwriting have been successfully implemented as off-the-shelf products, *offline*, unconstrained handwriting recognition has achieved much limited success. To date, the most notable successes of offline handwriting recognition are in digits recognition, with applications in postal address reading [Srihari and Kuebert, 1997] and bank check processing [Impedovo et al., 1997].

The relative success in digit recognition can be attributed to the availability of large training sets (with thousands of examples per digit), as well as carefully designed feature extraction procedures. For example, some of the best results (rivaling that of a human) reported for the MNIST digits database are achieved using convolutional neural network with architectures specifically designed for handwritten digits [Lecun et al., 1998; Simard et al., 2003; Ranzato et al., 2006]. We also see similar level of performance achieved using Support Vector Machines with jittered support vectors [Decoste and Schölkopf, 2002], where prior knowledge in the form of invariant transformations has been utilized.

In contrast to digits and Roman alphabets, offline handwritten Chinese character recognition remains a challenging problem after more than 20 years of research. Several factors contribute to this difficulty:

- Large set of characters – where the commonly used set contains more than 4000 characters, compared to the relatively small set of 26 Roman alphabets. This often results in limited available training data, where a typical database may contain as few as 200 examples per character.
- Each character, on average, has more strokes than a typical digit or Roman alphabet and therefore more complex structure and patterns.
- Large number of mutually similar pairs/sets of characters.

The large variety of patterns, and the existence of mutually similar pairs of characters result in potentially very large within-class variability but very small between-class difference among the examples, i.e. very low signal-to-noise ratio. Figure 1.1 shows some of these pairs.

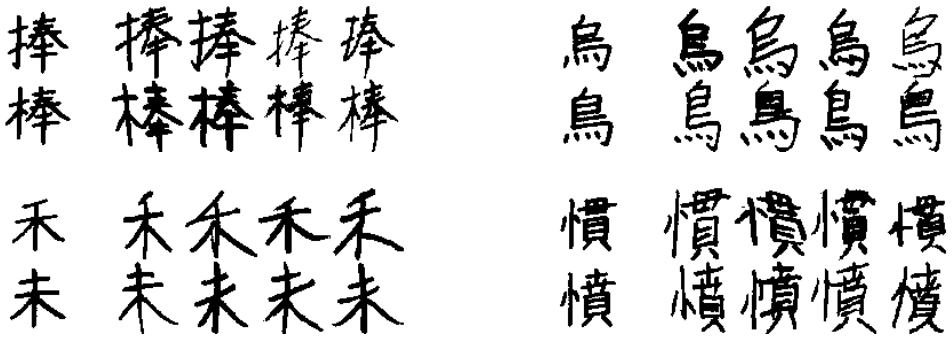


Figure 1.1: Some pairs of very similar characters

To a machine learner that sees only the pixel values within each example image, discovering the subtle differences between these pairs, if possible at all, requires an enormous amount of training data. One traditional approach to addressing this problem is to directly extract the underlying strokes of a given input and make decision based on stroke-level information. The problem with this approach is that stroke extraction is itself an inefficient and noisy process, especially in the presence of mutually similar characters.

We see that in the case of digits, even with ample training data, some relatively simple prior knowledge (e.g. invariant transformations) goes a long way to improving the learning performance.

It is natural to believe that prior knowledge should play even bigger roles in the case of Chinese characters.

We wish to incorporate our knowledge about strokes and interactions among strokes into the learning system during training. At the same time, we would like the final classifier to be efficient and robust in the presence of noise.

1.4 Our Proposed Approaches

We propose three approaches of incorporating prior knowledge into classification learning systems through model and feature construction. Each can be seen as a way of incorporating generative prior knowledge into a discriminative learner, with different levels of interaction between the feature construction process and the discriminative learner itself. EBL plays a central role of connecting the observable features (e.g. pixels) in the training examples to the abstract structure of strokes and stroke-level interactions.

Our first and simplest approach involves automated construction of generative models at a largely conceptual level (e.g. character strokes). These models, tuned with the actual training examples, are then used to generate artificial training examples, which we call *phantom examples*. The augmented training set, containing both real and phantom examples, can then be used as input to any discriminative learner to learn the final classifier.

While relatively simple to implement, the phantom examples approach does not fully utilize our prior knowledge about potential similarities and differences between characters. It also places much of the burden of learning on the discriminative learner. Both the second and the third approach employ the concept of *semantic features*, which are basically features that are more in line with a domain expert's vocabulary. For example, these are features that are derived from stroke-level interaction, in contrast to pixel-level interactions.

Our second approach constructs relatively simple semantic features based on stroke-level similarities and differences between characters. Similarities are exploited to produce perceptually salient *reference features* that can be reliably detected without knowing the label of the input. This serves as a mechanism of adaptively registering an input image. On the other hand, differences between characters are used to define *target features*, which correspond to regions of an image with high concentration of discriminative information. Figure 1.2 illustrates this. A mapping from reference to target features is learned from the training examples and the final classifier is

learned using the target features as input to a discriminative learner.

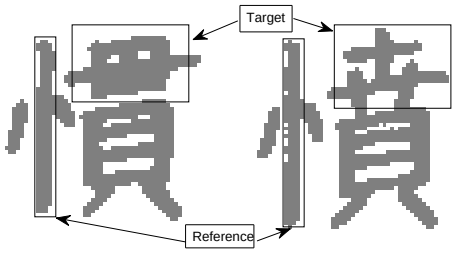


Figure 1.2: Example reference and target features

There exist situations when simple reference features are insufficient to accurately locate the target features. The third approach addresses this by constructing more complex semantic features, where the target feature in a given input is found in a procedural manner via a construct we call *sensor trees*. Each sensor tree can be thought of as encapsulating a complex reference feature that accumulates information from multiple sensors in order to achieve higher robustness and accuracy in the detection of the target feature. Furthermore, each sensor tree is optimized discriminatively and therefore can itself be the final classifier.

1.5 Scope

In this work, we use the tasks of distinguishing Chinese characters as the main illustrative domain. Furthermore, we restrict ourselves to classifying pre-segmented characters. However, the concept of semantic features, and the use of perceptually salient reference features to focus the attention of a classifier onto informative target features can be adapted to a larger scale, document-level recognition system, which can include coarser-grained layout analysis.

At the larger scale of document-level recognition system, one could utilize other sources of information, such as a language model. Although not necessarily applicable in all situations (e.g. isolated entries in a form), such models can potentially improve the overall recognition accuracy. We do not pursue this direction in this work, but it can be used in conjunction to our proposed methods.

By employing a different domain theory, the proposed approaches can be applied to domains beyond that of Chinese character recognition, to a much broader range of domains including object recognition or even non-image-based classification tasks. For closely related domains such as

handwriting in other languages (especially Asian scripts) or line drawings, our domain theory on Chinese characters can be easily transfered and adapted.

1.6 Organization

Chapter 2 provides an overview of the underlying principle for our approaches with regard to supervised learning and generalization.

Chapter 3 discusses the approach of model construction for phantom examples.

Chapter 4 and 5 describe the simple and complex semantic feature construction algorithms, respectively.

Chapter 6 provides a unifying view as well as comparison of the three approaches under various perspectives, particularly with regard to the use of generative and discriminative knowledge in learning.

Chapter 7 discusses related works in the literature.

The final chapter summarizes the contributions of this work and discusses potential future works.

Chapter 2

Working for the Right Reason: An Overview

2.1 Motivation

It is well-known that successful application of machine learning depends strongly on the kind of inductive bias that we impose on the learner. A very general kind of inductive bias involves a preference over the space of all possible solutions. We will be looking at preference in the form of a Bayesian prior over the solution space and analyze how this is related to our prior knowledge about the problem, especially in the context of feature construction. This analysis provides the motivation for our approaches to incorporating prior domain knowledge via explanation-based feature construction.

2.1.1 Classification Tasks and Solution Space

We first define what we mean by a *classification task*, or simply a *task*. A task is defined by the following components $\langle \mathcal{X}, \mathcal{Y}, D, l, \tau \rangle$, which include:

- An input space $\mathcal{X}$. We assume $\mathcal{X} = \mathbb{R}^n$ unless otherwise stated.
- An output space (i.e. the set of class labels) $\mathcal{Y}$. We assume $\mathcal{Y} = \{-1, 1\}$ unless otherwise stated.
- A distribution D over $(x, y) \in \mathcal{X} \times \mathcal{Y}$.
- A loss function $l : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$. A common loss function is the zero-one loss, where

$$l(y, y') = \begin{cases} 1 & \text{if } y \neq y' \\ 0 & \text{if } y = y' \end{cases}$$

- A performance threshold $\tau \in \mathbb{R}$, where $\tau \geq 0$.

For a given function $\hat{f} : \mathcal{X} \to \mathcal{Y}$, we define its loss with respect to D as $E_D(l(\hat{f}(X), Y))$ where E_D denotes the expectation with respect to D. We also consider stochastic solutions (Gibbs strategies) where for each new input, we pick a function according to a distribution Q over the function space ($\mathcal{Y}^{\mathcal{X}}$) and use it to label the input. The loss of this strategy is given by $E_{f\sim Q}E_D(l(f(X), Y))$. Note that a deterministic solution is simply a special case of Q where all the probability mass concentrates on one function. We use the shorthand $l(Q, D)$ to mean $E_{f\sim Q}E_D(l(f(X), Y))$.

A strategy Q is a solution for the task if $l(Q, D) \leq \tau$. It is possible that a solution does not exist – this happens when the minimal achievable loss, given by

$$l^* = \inf_{Q\in\mathcal{P}(\mathcal{Y}^{\mathcal{X}})} l(Q, D) \qquad ,$$

is greater than τ, where $\mathcal{P}(\mathcal{Y}^{\mathcal{X}})$ denotes the space of all strategies (i.e. the solution space). In the case of zero-one loss, l^* is given by the optimal Bayes error. Note that there exists a deterministic solution that attains the Bayes error, namely, by always choosing the label with the highest conditional probability $Pr(y|x)$. This of course requires that we know the distribution D.

2.1.2 Prior Knowledge and Training Data: An Information Point of View

Given a task $\langle \mathcal{X}, \mathcal{Y}, D, l, \tau \rangle$, we would like to find a solution by learning from a set of labeled examples (i.e. the training set) $\mathbf{z} = ((x_1, y_1), \ldots, (x_m, y_m))$, drawn i.i.d from D. The questions of interest here are:

1. How much information do we need to find a solution?
2. How much information does the training set provide?
3. How much prior information do we have about the task?
4. How much prior information can actually be utilized by the learner?

It is sometimes possible to quantify the information needed to find a solution. For example, when $|\mathcal{X}|$ is finite, the space of all mappings $\mathcal{Y}^{\mathcal{X}}$ is also finite where $|\mathcal{Y}^{\mathcal{X}}| = 2^{|\mathcal{X}|}$. Assuming that there is only one (deterministic) feasible solution $f^* \in \mathcal{Y}^{\mathcal{X}}$. Using Shannon entropy, the (maximally uninformative) uncertainty over the function space is $\log_2(2^{|\mathcal{X}|}) = |\mathcal{X}|$ bits. This is the amount of information we need to find f^*. Assuming that by "information in the training examples" we mean the information from the labels alone. In this case each example can contribute a

maximum of 1 bit, and we essentially need to see all possible $x \in \mathcal{X}$ to exactly pinpoint f^*. This is consistent with the no-free-lunch theorem.

In general, we usually have prior information in one way or another. For example, if it is known beforehand that $f^* \in \mathbb{H} \subset \mathcal{Y}^{\mathcal{X}}$ and $|\mathbb{H}| \ll 2^{|\mathcal{X}|}$, then we may not need to see all of $\mathcal{X}$ to find f^*.

Even when $\mathcal{X}$ is infinite (or even uncountable) and there is no prior information, it is still possible that the training set alone provides all the information we need to find a solution. For example, if training data is unlimited, we can use classification strategies such as k-nearest neighbor that are proven to be universally consistent (i.e. asymptotically optimal) under mild technical conditions.

On the other hand, when we have perfect knowledge about the task, no learning is necessary since we can directly identify and implement the solution.

The last question above is particularly interesting. While most learners can readily utilize the information in the training examples (i.e. information from the labels), it is unclear how the prior knowledge is utilized in terms of amount of information. Furthermore, there is a distinction between the prior knowledge that we do have (or can be readily obtained) and the prior knowledge that is actually utilized by the learner.

Since most practical problems fall somewhere in between the two extremes, it is helpful to know how much information from our prior knowledge can be utilized by the learner. In cases where the available training data is limited, this becomes crucial.

The PAC-Bayesian bound [Mcallester, 1999, 2003] allows us to investigate the above questions, at least approximately. Let P be a prior belief over the function space (P itself can be thought of as a strategy). The PAC-Bayesian bound is an upper bound to the generalization error of a strategy Q based on the KL-divergence between Q (the data-dependent "posterior") and the prior P. Let $\mathbf{z}$ be a random training set of size m, and $\hat{l}(Q, \mathbf{z})$ be the loss of strategy Q with respect to the (empirical) distribution induced by $\mathbf{z}$. With probability at least $1 - \delta$ over drawings of $\mathbf{z}$, we have

$$\forall Q \quad l(Q, D) \leq \hat{l}(Q, \mathbf{z}) + \sqrt{\frac{KL(Q||P) + \ln \frac{m}{\delta} + 2}{2m - 1}} \qquad . \tag{2.1}$$

In theory, we can find a solution with high probability by simply finding a Q such that $\hat{l}(Q, \mathbf{z}) + \sqrt{\frac{KL(Q||P) + \ln \frac{m}{\delta} + 2}{2m-1}} \leq \tau$.

Notice that $KL(Q||P)$ is the only term concerning the prior information. It is unclear how the prior P can be obtained, especially when the available prior knowledge is not in this form. For now, we assume that all our prior information does indeed go into P.

If we ignore the non-dominant terms and all the constants, the bound becomes

$$\hat{l}(Q, \mathbf{z}) + \sqrt{\frac{KL(Q||P)}{m}} \leq \tau \quad . \tag{2.2}$$

Fixing the sample size m, we see a tradeoff between the fit to the training data $\hat{l}(Q, \mathbf{z})$ and respecting the prior information $KL(Q||P)$. For any Q that satisfies the bound (i.e. Q a feasible solution), and when $\hat{l}(Q, \mathbf{z})$ is large (i.e. Q does not fit the data well), $KL(Q||P)$ must be small in order for Q to satisfy the bound. This means that the information that we use to convince ourselves that Q is a solution comes from the fact that it very much respects the prior P. We can therefore use the quantity $\frac{1}{KL(Q||P)}$ as a measure of the amount of prior information actually utilized by Q, given that Q satisfies the bound. We denote this as $\Delta(Q, P) = \frac{1}{KL(Q||P)}$.

Note that our interpretation of $\Delta(Q, P)$ is more qualitative than quantitative. Nevertheless it provides a basis in which to measure the amount of prior information actually utilized by a given solution.

Rearranging terms in 2.2, and plugging in $\Delta(Q, P)$, we obtain:

$$m \geq \Big(\frac{1}{(\tau - \hat{l}(Q, \mathbf{z}))^2}\Big)\frac{1}{\Delta(Q, P)} \quad .$$

Since any feasible solution Q must also satisfy $\hat{l}(Q, \mathbf{z}) \leq \tau$, we have

$$m \geq \Big(\frac{1}{\tau^2}\Big)\frac{1}{\Delta(Q, P)} \quad .$$

Fixing τ, we observe the tradeoff between the data (characterized by the sample size m) and the prior information (characterized by $\Delta(Q, P)$). Figure 2.1 shows this tradeoff. All solutions that satisfy the bound must belong in the shaded region.

Note that the PAC-Bayesian bound is just an upper bound. In other words, there may exist Q that is a feasible solution for the task but violates the bound when applied to certain training sets. We define the amount of prior information in a specific P that can be utilized in solving the *task*, independent of any training data, based on the true loss of any Q (i.e. $l(Q, D)$) as:

$$\Delta(P) = \sup_{Q: l(Q,D) \leq \tau} \Delta(Q, P)$$

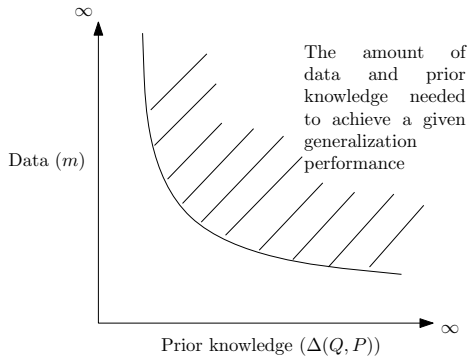


Figure 2.1: Tradeoff between the amount of prior knowledge and training data needed to achieve a specific generalization performance

Figure 2.2 shows an example solution space where the shaded regions are the solutions with true loss $l(Q, D) \leq \tau$. The solution that maximally utilizes the prior information is marked as Q'. Note that the KL-divergence not a distance metric so the only distance that is meaningful is between each Q and the prior P (i.e. $KL(Q||P)$). Also note that Q' is not necessarily the optimal solution.

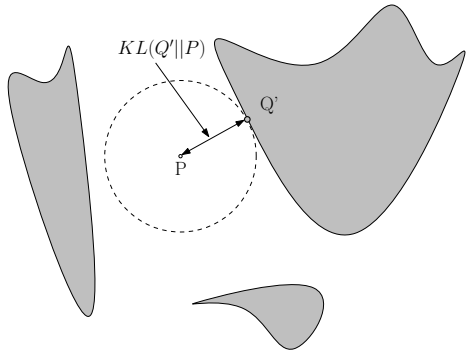


Figure 2.2: Solution space. Q' utilizes the most information from P.

$\Delta(P)$ cannot be easily computed in practice since it depends on the true loss $l(Q, D)$ and requires a maximization over all strategies Q. However, if D is known, we can approximate $\Delta(P)$ using only the solutions that are actually found by the learner. This will be illustrated in the next section.

2.1.3 Features and Prior Knowledge: An Illustration

There exist real-world problems where adding more data is no longer an option. This could be due to limitations in computational/storage resource, or the fact that it is too costly to obtain

more data. In this case, incorporating more prior knowledge into the learning system becomes crucial, and possibly the only option. However, in contrast to adding more data, incorporating more "knowledge" can be a challenging problem, which is still largely unsolved.

While there are many ways of incorporating prior knowledge into the learning system, we believe that the most fundamental is through the construction of good models. For classification problems this usually translates to constructing good features. The learner itself can often become inconsequential once the "right" models are constructed. Model construction (as opposed to model selection), however, is rarely done automatically and is sometimes considered an "art". Tools such as Bayesian inference or MDL can be powerful in selecting among alternative models but they do not tell us how to come up with the models in the first place. SVM learning, for example, involves the choice of a good kernel, which can be seen as a form of model construction, where features are implicitly constructed.

Equipped with the definition from the previous section for the contribution of prior information to a task, we illustrate our point with a simple but non-trivial real-world classification task. In particular, we wish to evaluate the amount of information utilized by the learning system when different kinds of features are used in learning. Ultimately, we seek evidence that this amount of information directly affects the generalization performance.

We shall view the learning task as consisting of two stages.

- The first stage involves feature construction. The outcome of this stage is a feature transformation that will be applied to every task input.
- The second stage involves finding a classifier from a class of functions on the (transformed) input features. This is usually named the hypothesis space. We will restrict ourselves to only linear functions.

We will employ the SVM as the learner in the second stage. The use of kernels in SVM allows an implicit transformation of every input to a possibly higher dimensional feature space. This can itself be viewed as the first stage, or an extension to the first stage.

We assume that the prior P assigns uniform probability to every function in the hypothesis space and zero elsewhere, where "elsewhere" refers to all other functions not in the hypothesis space. For the SVM the hypothesis space consists of linear functions on the kernel-induced feature space.

The inputs of this task are images of single handwritten strokes, extracted from actual hand-

written Chinese characters. The task is to differentiate between images containing horizontal strokes from those with vertical strokes. Figure 2.3 shows some examples.

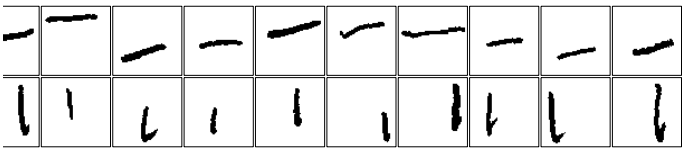


Figure 2.3: Example images of horizontal (top row) and vertical (bottom row) strokes

Each image is a bitmap of 64-by-64 pixels. We added translations and slight rotations in each example to simulate the variations in locations and orientations of single strokes relative to the other strokes in actual, multi-stroke characters.

We will evaluate three types of features:

1. A baseline, "knowledge-poor" approach to this problem is to simply use the raw input features, i.e. to represent each image as a 4096-dimensional vector of pixel values. One could then employ various types of kernels to build nonlinear features over the raw inputs. We note that this transfers the feature construction problem to a kernel construction problem.

2. For the second, more "knowledge-rich" set of features, we employ the weighted directional histogram (WDH), which is known to be particularly useful for classifying handwritten Chinese characters. Each example is transformed into a 392-dimensional vector summarizing edge directions over different regions of the input image.

3. The third set of features, which is very task specific, is based on the knowledge that each stroke can be approximated by a straight line segment, and that the Hough transform can be used to extract line segments from an image. We perform a Hough transform on each input image and extract the longest detected line segment. The positions of the center and the endpoints, as well as the length and angle of the extracted line segment are then used to construct an 8-dimensional feature vector representing each example. We will call this the Hough feature.

For each feature set, we perform SVM learning with linear, polynomial as well as RBF kernels. SVM parameters (e.g. polynomial degrees) are optimized during training with grid search based

on cross-validation error estimates. A separate test set is used to estimate the test error (we assume zero-one loss in this task). Figure 2.4 shows the results.

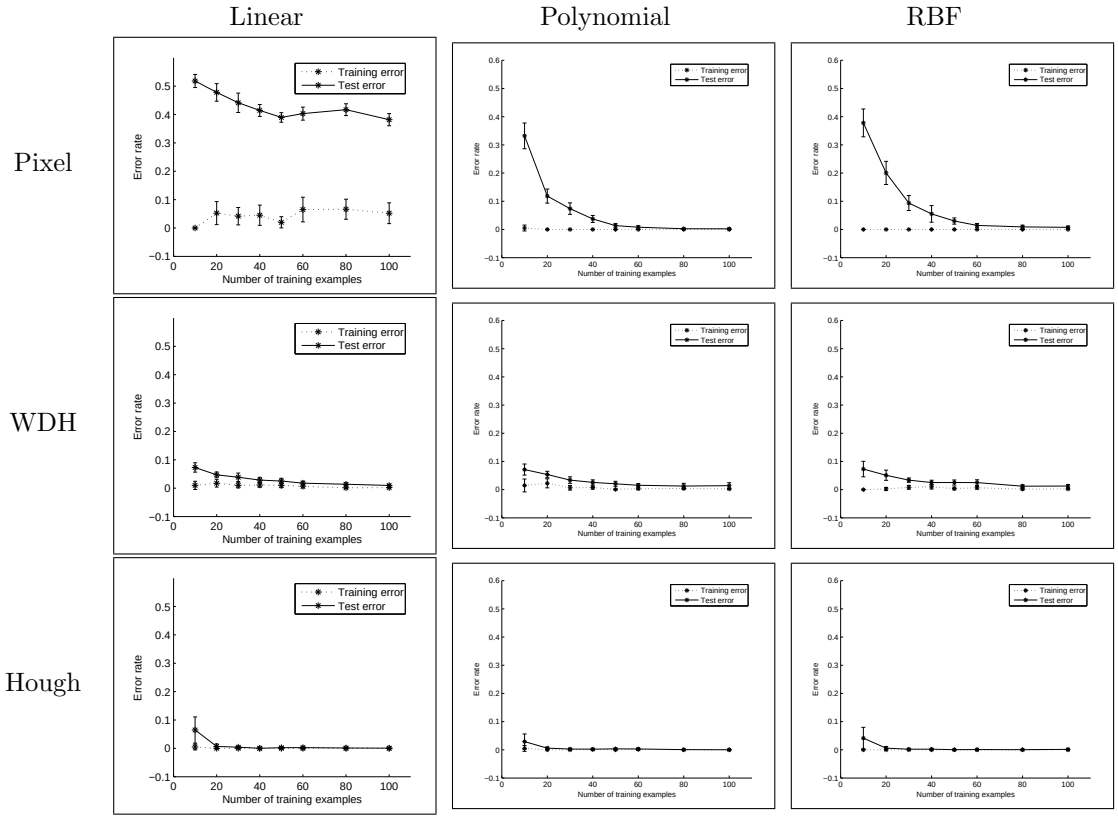


Figure 2.4: SVM learning results

With the raw pixel representation, we observe a large performance improvement when moving from linear to the polynomial and RBF kernels. While the raw pixel representation retains all the information in the input, the information crucial to the classification task (i.e. the stroke angle in this case) is "hidden" in a nonlinear manner within the pixel values. (Note: this "information" is between $\mathcal{X}$ and $\mathcal{Y}$ and should not be confused with the information for the learning task itself.)

With both the WDH and the Hough features, we do not see significant difference in generalization performance when we move from linear to the polynomial and RBF kernels. This suggest that the high dimensional feature space afforded by the polynomial or the RBF kernel is not necessary when the nonlinearity (in this case, from pixel values to line directions) is already captured in the feature representation of the input images. This suggests that once the key nonlinearity (or the relevant latent feature) is revealed, no significant performance gain can be achieved except by adding more data.

We assume the performance threshold $\tau = 2.5\%$. The number of training examples required to achieve a test error of less than 2.5% is around 40 for the pixel features, 20 for WDH, and less than 10 for the Hough features. These are the minimum number of examples needed in each case to achieve the specified performance. In other words, these solutions are the most likely to maximally utilize the information in the prior P in each respective case. With regard to Figure 2.1, they should be close to the boundary of the shaded region, and we would like to estimate the value of $\Delta(P)$ in each of the three cases.

The definition of $\Delta(P)$ involves finding a solution with the smallest $KL(Q||P)$. We would approximate this with the maximum margin solution found by the SVM. Denoting the maximum margin linear function $f_{\mathbf{w}}$ where $\mathbf{w}$ is the weights in the feature space, using the idea of Bayes admissibility [Herbrich and Graepel, 2001], this solution has the same loss as a uniformly distributed Gibbs strategy over a hypersphere centered on $f_{\mathbf{w}}$ with a radius based on the margin. Let $Q_{\mathbf{w}}$ denote this strategy, then $KL(Q_{\mathbf{w}}||P)$ can be approximated using the log-ratio between the volume of the sphere (V_Q) and the volume of the support of P (V_P).

Let $\Upsilon(\mathbf{w})$ be the normalized margin of $f_{\mathbf{w}}$ with respect to the training set, defined by

$$\Upsilon(\mathbf{w}) = \min_{i=1,\ldots,m} \frac{y_i \langle \varphi(x_i), \mathbf{w} \rangle}{||\mathbf{w}|| \cdot ||\varphi(x_i)||} \quad ,$$

where $\varphi(x_i)$ is the feature vector of input x_i in the kernel-induced feature space. Then $KL(Q||P)$ is given by [Herbrich and Graepel, 2001]:

$$KL(Q_{\mathbf{w}}||P) = \ln\left(\frac{V_P}{V_Q}\right) \leq d \ln\left(\frac{1}{1 - \sqrt{1 - \Upsilon^2(\mathbf{w})}}\right) + \ln(2)$$

where $d = \min(m, n_\varphi)$ (n_φ is the dimension of the feature space, which can be infinite as in the case of the RBF kernel).

From the margin, we obtain an upper-bound for $KL(Q_{\mathbf{w}}||P)$, and therefore a lower-bound for $\Delta(Q_{\mathbf{w}}, P)$. Using the actual observed margin of the learned solution from the training data, we obtain Figure 2.5. As approximations for $\Delta(P)$ these certainly are not accurate (they are simply lower bounds), but the qualitative pattern as expected in the previous discussion is evident.

Note that each point in Figure 2.5 comes from a solution based on different prior (P). They all achieve roughly the same true loss (as estimated by the test set). The knowledge-rich features definitely show that the induced hypothesis space allows more information to be utilized by the learner, and therefore needs less training data to achieve the same performance. Interpreting the

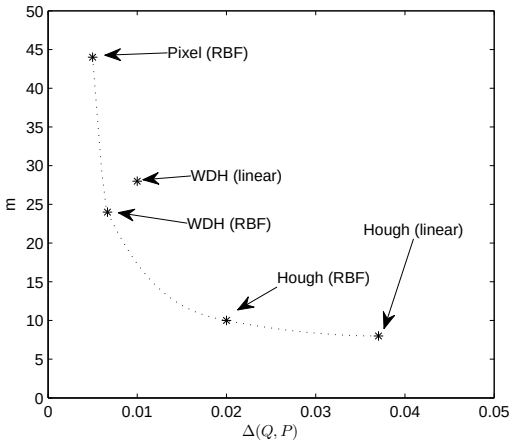


Figure 2.5: Data-Knowledge Tradeoff

features as Bayesian priors allows us to gain insight into their contributions in terms of information used in learning.

2.2 Working for the Right Reason

The idea of feature construction as an implicit prior, together with the PAC-Bayesian bound, can be used as a basis for a principled approach to learning classifiers that generalize well.

Recall Eq. 2.1. Given a fixed prior P, and a fixed training set with size m, all feasible solutions Q for a given task must satisfy

$$KL(Q||P) \leq \tau^2(2m-1) - \ln\frac{m}{\delta} - 2 \quad .$$

We will not be concerned with the actual numerical values of this constraint but rather the fact that all strategies that deviate too much from the prior need not ever be examined by the learner. On the other hand, any strategies within this bound that *work* well on the training data (i.e. small $\hat{l}(Q, \mathbf{z})$) will automatically generalize well.

An optimal learner will only search for solutions within this "feasible" region – these are the solutions that work for the right reason according to P. This is illustrated in Figure 2.6.

Searching for solutions within the feasible region can be realized and automated if we can come up with a structure that characterizes all the solutions that are close to P.

A simple example of such structure is to partition the entire function space into an ordered

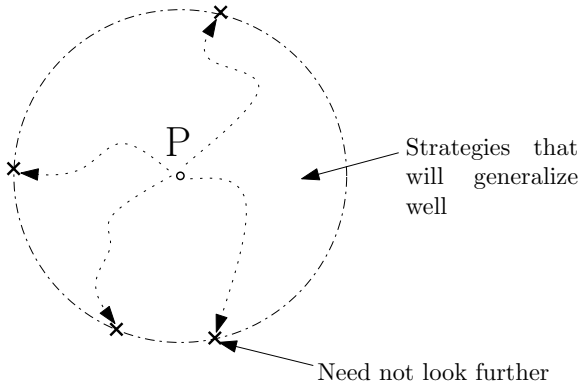


Figure 2.6: Feasible Solution Space

sequence of subsets $(\mathcal{F}_1, \mathcal{F}_2, \ldots)$. Each subset is then assigned a probability mass in an exponentially decaying manner. Equivalently, we can provide an encoding of each subset. If we use a proper prefix code, the probability for each subset is then given by Kraft's inequality based on the length of the code, where longer codes have smaller probability. The probability mass of each subset can then be distributed uniformly to all its members.

Given such a partition, we can find a feasible solution by finding a solution within each partition, starting with the one having the shortest code. Assuming that we use binary codes, and that the solution Q has uniform distribution over a volume V_Q within partition $\mathcal{F}$. Let $V_{\mathcal{F}}$ be the volume of $\mathcal{F}$ and that L is the code length of $\mathcal{F}$. $KL(Q||P)$ is then given by

$$\begin{aligned} KL(Q||P) &= \int_Q \frac{1}{V_Q} \ln \frac{2^L V_{\mathcal{F}}}{V_Q} \mathrm{d}\mathcal{F} \\ &= L(\ln 2) + \ln \frac{V_{\mathcal{F}}}{V_Q} \\ &= L(\ln 2) + KL(Q||P_{\mathcal{F}}) \end{aligned}$$

where $P_{\mathcal{F}}$ denotes the prior P restricted to $\mathcal{F}$.

We see that there is a price $L \ln 2$ for solutions in partitions with code length L. At certain point L will be too large for $KL(Q||P)$ to remain within the bound and the search can be stopped.

A brute-force search within the feasible space may still be computationally infeasible. We will see that information within the training examples, other than the label itself, can be exploited to facilitate this search.

2.3 Domain Knowledge and EBL

We use the term *domain knowledge* to refer to general understanding of how the world works, from an expert's point of view. When this knowledge has been precisely encoded in an appropriate vocabulary, the resulting collection of information will be referred to as the *domain theory*.

Expert-supplied domain theory has the strength that it can provide information that can otherwise be very hard to obtain statistically, e.g., the concept of a "line" and "angle" in an image. However, with regard to any specific real-world task, we generally do not place belief in any part of the domain theory itself unless there is evidence that supports such belief. We rely on real examples to supply such evidence.

Explanation-Based Learning (EBL) can be viewed as a learning paradigm where actual training examples are used to extract task-relevant information from an expert-supplied domain theory. This is achieved through *explanation* of the training examples. Being able to explain the training examples is the evidence for the relevance of the particular part of domain theory with respect to the task.

Statistical inference techniques can be powerful and robust but typically rely on the availability of good models. Bayesian inference, for example, is an excellent model selection tool but it does not tell us *how* to construct models. We can think of the domain theory as a model generator. Tools such as Bayesian inference can then be utilized by the explanation process to assess how well the models fit the real world data.

We use the term *model* to refer to either an abstract description of objects or a collection of distributions over objects (or both). We use the term *feature* to refer to either a function on the raw input or the output of such functions.

2.3.1 Semantic Features

Given a label, a concept, or an object category, a domain expert can usually provide information that exposes further structure in the concept or class of objects. For example, parts that constitute an object can be identified, together with the relationship between parts.

However, such information is often expressed in a vocabulary that is different from what is available in the raw input. For example, a handwritten character can be easily described in terms of the strokes that form the character, but strokes are not directly observable if the inputs are images.

If strokes live in a space $\mathcal{S}$ while the input images live in $\mathcal{X}$ then knowing the relationship between $\mathcal{S}$ and $\mathcal{X}$ allows us to exploit the structure in $\mathcal{S}$ in solving our tasks. We use the term *semantic features* or *high-level feature* to refer to features on $\mathcal{S}$. This definition is certainly relative, and in general there can be a hierarchy of features with multiple levels of abstractions. In this sense, the class label itself can be viewed as the highest level feature.

If the domain theory supplies a model that connects semantic features to the input $\mathcal{X}$, then we can infer the hidden features from the observed inputs. These models can be logical, probabilistic, or a mixture of different models based on the specific domain theory. We use the term *explanation* to refer to the process of inferring higher-level features from the observables. In the case of a logical model, the explanation process may involve theorem proving. In the case of a probabilistic model, the explanation process involves probabilistic inference (e.g. Bayesian inference).

Notice that generalization bounds such as the PAC-Bayesian bound only utilizes the label ($\mathcal{Y}$) information (i.e. between $\mathcal{X}$ and $\mathcal{Y}$, characterized by the empirical loss). Semantic features allow the learning system to obtain additional information within the input $\mathcal{X}$ itself. Turning this information into a form usable by the learning system is the key challenge that we address in this work.

2.3.2 Well-formed Concepts

Semantic features impose both a structure over the input space $\mathcal{X}$ and a restriction over the solution space. For example, most images cannot legitimately contain a handwritten character, and that a horizontal stroke should be distinguished from a vertical one based on its angle.

We can think of the space of solutions when all the inputs are given in terms of the semantic features. We use the term *well-formed concepts* to refer to solutions in this space. When a solution that corresponds to a well-formed concept actually performs well on the training example, we have strong evidence that it works for the right reason, and therefore should generalize well.

A direct, but potentially costly approach to learn well-formed concept is to perform inference on every input to reveal the hidden features. For example, a stroke extraction procedure can be applied to every input image to extract stroke configurations. The extracted strokes can then be used to make decision about the class label. This can be computationally costly and the stroke extraction procedure may also be vulnerable to noise if not properly designed.

Instead, EBL approaches focus on spending the computational resources during training, but learn efficient and robust feature detectors to be used in the final solution. This is depicted in Fig-

ure 2.7.

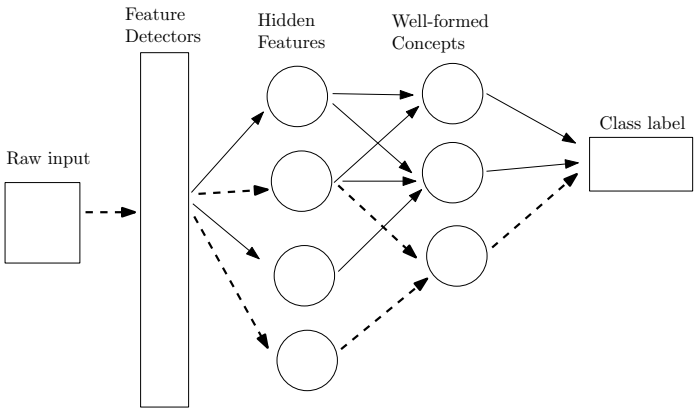


Figure 2.7: Semantic/High-level features

2.3.3 Well-Formed Concepts and Bayesian Prior

With well-formed concepts, we now have a principled way to connect our domain theory to the Bayesian prior P. In particular, each well-formed concept is associated to a partition over the function space as described in section 2.2. From the coding point of view, this also suggests that simple concepts are associated with shorter codes, and therefore are always explored first compared to more complex concepts.

Notice that we utilize the training examples in two different ways. First, the information in both $\mathcal{X}$ and $\mathcal{Y}$ is used in the explanation process to reveal the hidden features and the space of well-formed concepts. Learning the final classifier can then proceed via robust statistical techniques and this mainly utilizes only the label information (i.e. to find solution that has low empirical loss).

Chapter 3

Model Construction for Phantom Examples

3.1 Introduction

Combining generative and discriminative approaches to statistical learning has been an important research direction in machine learning and pattern recognition [Tong and Koller, 2000; Lin et al., 2005; Jaakkola and Haussler, 1999; Raina et al., 2004]. The generative approach models the data generation process explicitly, usually in a top-down manner, e.g. modeling $Pr(X|Y)$. This is often consistent with the natural way we describe objects. For discriminative tasks (e.g. classification) however, the generative approach may perform poorly when the models are inadequate.

Discriminative learners model the decision boundary directly, e.g. by modeling $Pr(Y|X)$ directly, and can outperform generative learners when the training size is large [Ng and Jordan, 2002]. However, it is usually much harder to incorporate prior knowledge into the learning system.

We propose the use of artificial training examples, which we call *phantom examples* as a mean to incorporate generative prior knowledge into a discriminative learner.

Artificial, or "virtual" training examples have been used in learning systems under different settings. [Baird, 1992] uses a document image defect models to create new examples by subjecting the original examples to various noise or defects. [Poggio and Vetter, 1992] uses known symmetries and transformations to create new examples that are new "views" of the original examples. Invariant transformations are also used to create "virtual support vectors" [Burges and Schölkopf, 1996; Decoste and Schölkopf, 2002].

These works make use known transformations that are applied to the input examples in order to generate new ones. These transformations are known to preserve the label or class-membership of the examples and are predefined for each specific task. While easy to apply, such transformations do not task advantage of extra information within the content of each input example that can be helpful to the learning task. In order words, there is no interaction between the training examples and prior domain knowledge involved.

[Miyao and Maruyama, 2006] makes use of online strokes information of handwritten characters and learns a generative model for Chinese characters at the strokes level. Its motivation was that the use of stroke-level modeling can capture more of the variability in handwritten Chinese characters. The "hidden", stroke-level information is provided by online examples. There is interaction between the training examples and the prior knowledge in terms of fitting the strokes distribution to a predefined model.

Our proposed phantom examples approach can be seen as a generalization of the above approaches. The invariant transformations can be seen as a generative model where each task instance is generated by a random transformation applied to a prototype, which is in turn drawn from some underlying distribution.

We use the term "model" to refer to either a particular probabilistic source, or a collection of (maybe parametric) probabilistic sources, depending on the context.

In most real-world tasks the range of possible models will be huge. Take handwritten character images for example, the final image may be affected by the writing style of the writer, the writing implement, the digitization noise, and even the mental state of the writer. Most of these factors would be unobservable, but appreciation of such factors may substantially benefit the learning system.

We would like to build a system that admits an expressive space of possible models, and learn an appropriate model after looking at the actual training examples. The key to successful learning in such space is that only aspects of the domain theory that are actually "observed" in the training examples will be modeled. The learned model is then used to generate the phantom examples, which are added to the training set for subsequent discriminative learning.

In the ideal case where the true model is known, the learning problem becomes easy. Since, in principle, we could use the model to generate an arbitrarily large training set and as long as a consistent learner is used, we can allow an arbitrarily rich hypothesis space.

In practice, we face the following three challenges:

1. How do we deal with unobservable, conceptual objects or "hidden features"?

2. A domain theory may suggest many (or even infinitely many) potential models, how do we find an adequate one?

3. It is almost certain that our models are wrong. What is the risk when we employ phantom examples that are generated from such models?

We associate the first challenge with the process of "explaining" the training examples. An explanation of a training example with respect to its class label reveals the conceptual objects/features that, according to the domain theory, explains why the example qualifies as a member of that particular class. We note that the explanation process is meant for *labeled* training examples, and is therefore only needed during the training process. Explanations can either come from an existing information source (such as online examples), manually constructed by an expert, or extracted from the training examples by an automated algorithm – usually involving some form of inference (logical and/or probabilistic).

The second challenge is usually associated with the "model selection" problem. Consider the task of distinguishing a pair (or more generally, a set) of mutually-similar Chinese characters, where the differences among these characters can be very subtle. It is conceivable that each set of these similar characters needs a different model, tailored such that informative parts of the characters are modeled with more details. Conventional model selection techniques such as AIC [Akaike, 1974], BIC [Rissanen, 1978] and MDL [Schwarz, 1978] rely on an a priori defined space of models, where the models are evaluated against the training data in order to find a balance between good fit and model complexity. With these approaches, we would need to either predefine a huge set of possible models to accommodate all such sets, or hand-tailor a smaller, relevant set of models for each task. The former method requires the system to evaluate too many irrelevant models and risk overfitting to a wrong model, while the latter requires too much manual effort and is often impractical.

In order to work with a huge and expressive space of possible models, some structure within the space would be necessary. One common approach is to predefine a hierarchy of models, usually sorted from the simplest to the most complex. We would like to introduce a structure beyond that of a simple hierarchy and that does not require the system to evaluate all potential models. A suitably designed domain theory can supply the conceptual objects and relationships between such objects that in turn create a structure over the space of possible models.

We address this problem by employing an automated, adaptive strategy where an initial, simple (or trivial) model is refined in stages, and guided by observing how the current model "explains" the training data. Such system allows the possibility of employing a very expressive space of potential models, but for any particular task, only a small fraction of this space needs to be explored. We call this process *explanation-based model construction*.

The final challenge deals with the analytical properties of a system trained with phantom ex-

amples. Due to the use of incorrect models, it is no longer true that the larger the training set, the better the learning result. We will address this issue both analytically and empirically throughout this chapter.

3.2 Analysis

3.2.1 Domain Knowledge Bias

Consider a binary classification task where given the input $x \in \mathcal{X}$ we try to predict the label $y \in \{-1, 1\}$, assuming that there exists an underlying joint distribution D for $(x, y) \in \mathcal{X} \times \mathcal{Y}$ which is unknown. When training a discriminative learner, we present the learner with labeled examples from two different classes, sampled from their respective underlying true distributions $\mathbf{p}_0(X) = \Pr_D(X|Y = -1)$ and $\mathbf{p}_1(X) = \Pr_D(X|Y = 1)$.

We can think of communicating the distributions $\mathbf{p}_0(X)$ and $\mathbf{p}_1(X)$ to the learner through the training examples. However, for a finite training set, the empirical distribution may be inadequate for the learner to come up with a good decision boundary.

We augment the training set by generating phantom examples drawn from class-specific generative models with parameters $\theta = (\theta_0, \theta_1)$, where $\theta_0 \in \Theta_0$ describes the model for class -1 and $\theta_1 \in \Theta_1$ for class 1. The corresponding distributions are denoted $\mathbf{q}_0^{\theta_0}(X) = \Pr_{\theta_0}(X|Y = -1)$ and $\mathbf{q}_1^{\theta_1}(X) = \Pr_{\theta_1}(X|Y = 1)$ respectively. Both θ_0 and θ_1 are obtained by fitting the parameters to the real training set.

For complex, real-world problems, it is extremely unlikely that our space of generative models contains the true distributions $\mathbf{p}_0$ and $\mathbf{p}_1$. However, we expect that $\mathbf{q}_0$ and $\mathbf{q}_1$ will be closer to the true distributions when tuned with more real examples. Using KL-divergence as a measure of closeness, we define the *bias* of the trained generative models as follows:

$$Bias(\theta_0, \theta_1) = KL(\mathbf{p}_0||\mathbf{q}_0^{\theta_0}) + KL(\mathbf{p}_1||\mathbf{q}_1^{\theta_1})$$

where KL denotes the KL-divergence.

Since the KL-divergence is nonnegative, there exist a minimal bias

$$\epsilon_\theta^* = \inf_{\theta_0, \theta_1} Bias(\theta_0, \theta_1).$$

We call this the bias of our domain knowledge. It is reasonable to expect that, when trained with

more real examples, the resulting models θ_0 and θ_1 will have bias $\epsilon_{\theta_0,\theta_1}$ closer to ϵ^*_θ.

The bias provides an upper bound to the deviation from the optimal Bayes error rate when distributions $\mathbf{q}_0$ and $\mathbf{q}_1$ are used to classify test examples which are drawn according to $\mathbf{p}_0$ and $\mathbf{p}_1$. This is given by:

Proposition 1. *Assume that the class prior for each class is uniform (i.e. $Pr(Y = -1) = Pr(Y = 1) = 1/2$). Let ϵ^* be the optimal Bayes error for examples drawn according to $\mathbf{p}_0$ and $\mathbf{p}_1$. The expected classification error when $\mathbf{q}_0^{\theta_0}$ and $\mathbf{q}_1^{\theta_1}$ are used to make decision (instead of $\mathbf{p}_0$ and $\mathbf{p}_1$), denoted $\epsilon_{\theta_0,\theta_1}$, is upper-bounded by:*

$$\epsilon_{\theta_0,\theta_1} \leq \epsilon^* + \sqrt{\frac{Bias(\theta_0, \theta_1)}{2}} \tag{3.1}$$

Proof.

$$\begin{aligned}
\epsilon_{\theta_0,\theta_1} - \epsilon^* &\leq \sum_X \frac{1}{2}|\mathbf{p}_0(X) - \mathbf{q}_0(X)| + \frac{1}{2}|\mathbf{p}_1(X) - \mathbf{q}_1(X)| \\
&\leq \frac{1}{2}\Big(\sqrt{2KL(\mathbf{p}_0||\mathbf{q}_0)} + \sqrt{2KL(\mathbf{p}_1||\mathbf{q}_1)}\Big) \\
&\leq \sqrt{\frac{KL(\mathbf{p}_0||\mathbf{q}_0) + KL(\mathbf{p}_1||\mathbf{q}_1)}{2}}
\end{aligned}$$

where the first inequality is due to [Devroye et al., 1996] and the second inequality employs the Pinsker's inequality. □

Proposition 1 tells us that as long as the generative models are close enough to the true models, the achievable error rate will be close to the true optimal. Depending on the problem, it may or may not attain the true optimal, and this largely depends on the amount of information our generative models can provide regarding the *decision boundary* of the problem.

Assume that the space of classifiers learnable by our discriminative learner includes those with performance arbitrarily close to the optimal Bayes error. Let $\epsilon_{\tilde{N},M,N}$ be the expected error of the resulting classifier when N real examples and M phantom examples are used to train the discriminative learner, where the phantom examples are generated using a model tuned with $\tilde{N}$ real examples ($\tilde{N}$ and N will usually be the same, i.e. all real examples participate in tuning the generative models as well as training the final classifier). Let ϵ^* be the best possible error rate (Bayes optimal error) for the particular problem distribution D, we expect that $\epsilon_{\tilde{N},M,N} \to \epsilon^*$ as the number of real examples $N \to \infty$ (holding $\tilde{N}$ and M fixed), assuming that a consistent discriminative

learner is used.

The more interesting question would be regarding the expected performance when we increase the number of phantom examples used in the training of the discriminative learner. This largely depends on the quality of the generative models used, which in turn depends on the quality of the prior domain knowledge. Regardless of this, we expect that $\epsilon_{\tilde{N},M,N} \to \epsilon^*_\theta$ when both $\tilde{N}, M \to \infty$ (holding N fixed).

As the phantom set grows, we can expect that the amount of information that it extracts from the generative models to reach an asymptote (the corresponding error rate would be ϵ^*_θ). If the generative models are helpful (good domain knowledge), the effect will be positive; if the generative models are harmful (bad domain knowledge), the effect will be negative.

Another advantage of using phantom examples, in addition to the improved classification accuracy, is the robustness to classification noise. With the domain knowledge, the classifier is learned not only on the noisy distribution given by the training examples, but a distribution biased to one justified by the domain theory (e.g. the generative models for the hidden strokes), making the classifier less likely to fit the noise. When tested on the (noise-free) testing data, the classifier may more likely reveal the true labeling of the examples rather than the noisy labeling.

3.2.2 Model Space

Since we tune our generative models using only a finite number of real examples, it is susceptible to overfitting. This in turn will affect the quality of the actual phantom examples generated. We would like our system to adaptively select a model that achieves a balance between good fit and model complexity.

Instead of using an a priori fixed generative model for each class of objects, we allow a (possibly infinite) space $\mathcal{M}$ of different generative models, each with different complexity and can be parameterized differently. As mentioned earlier, learning in the space of models can be greatly facilitated by having a structure over the model space $\mathcal{M}$. The structure should ideally allow a local search within model classes, where models of similar complexities are close to each other. For this work, we assume that each object class can be decomposed into conceptual parts. The model space is then constructed based on the models for individual parts.

We first define the variables involved in any particular model. We use $\mathcal{X}$ to denote the input space (for example, images) and X the corresponding random variable that represents an input instance, where X takes values from $\mathcal{X}$. It is often the case that $\mathcal{X}$ is a high-dimensional space

where each individual component contains very little information about the class label (e.g. pixels in an image). Modeling a distribution over $\mathcal{X}$ is therefore undesirable. With prior domain knowledge, however, we can usually identify a higher-level, low-dimensional space of "hidden" variables (such as properties of strokes in a character), denoted here as Z, which takes values from $\mathcal{Z}$. We assume that there is a function $\phi : \mathcal{Z} \mapsto \mathcal{X}$ that converts each instance of Z into a corresponding input-space instance $X = \phi(Z)$.

The variable Z can be further decomposed as:

$$Z = (Z_{\Gamma_1}, Z_{\Gamma_2}, \ldots, Z_{\Gamma_k})$$

where each $\Gamma_i, (i = 1, \ldots, k)$ denotes a model for part i of the object class/category. Each model Γ_i is represented by a $d(\Gamma_i)$-dimensional random vector, which we denote as

$$Z_{\Gamma_i} = (Z^1_{\Gamma_i}, Z^2_{\Gamma_i}, \ldots, Z^{d(\Gamma_i)}_{\Gamma_i}), \quad i = 1, \ldots, k.$$

Therefore, Z is a $\tilde{d}$-dimensional random vector where $\tilde{d} = \sum_{i=1}^{k} d(\Gamma_i)$.

The model Γ_i for each object part i can be chosen differently from a set of possible models $\mathit{\Gamma}^*$. For example, when modeling strokes in a handwritten character, each stroke can either be modeled as a straight line segment or a curve segment. In this case, let $\mathit{\Gamma}^* = \{\mathit{\Gamma}_{line}, \mathit{\Gamma}_{curve}\}$. Then each Γ_i can either be $\mathit{\Gamma}_{line}$ or $\mathit{\Gamma}_{curve}$.

The complete generative model for a particular object class C that consists of k object parts can be fully specified by selecting an object model for each part. We use a k-tuple $\Gamma^C = \langle \Gamma_1, \ldots, \Gamma_k \rangle$ to represent this selection. Again, using the stroke models above, a character with 3 strokes, for example, can have a particular model $\langle \mathit{\Gamma}_{line}, \mathit{\Gamma}_{line}, \mathit{\Gamma}_{curve} \rangle$, indicating that the first 2 strokes are modeled as straight lines, while the 3rd stroke as a curve segment.

Considering now a binary classification task between some object classes C_0 and C_1. Let k_0 and k_1 denote the number of parts in each object class respectively, then the space of possible models for this particular task, $\mathcal{M}^{C_0,C_1}$, consists of all possible model combinations for C_0 and C_1, that is,

$$\mathcal{M}^{C_0,C_1} = \{\langle \Gamma^{C_0}, \Gamma^{C_1} \rangle\}$$

where Γ^{C_0} and Γ^{C_1} are k_0-tuple and k_1-tuple describing the respective choices of part models for each class.

By decomposing each object into parts and allowing the possibility of modeling each part differently, we have introduced a structure within the model space. By taking advantage of this structure, we can perform adaptive model refinement that does not require the system to evaluate every possible model and can therefore entertain a much richer space of models.

3.2.3 Model Space and Well-Formed Concepts

Each member of $\mathcal{M}^{C_0,C_1}$ (i.e. an instance of $\langle \Gamma^{C_0}, \Gamma^{C_1} \rangle$) can be seen as representing a *well-formed concept* discussed in Chapter 2. Each pair of distributions (over Z) for class C_0 and C_1 defines an implicit decision boundary, which corresponds to the Bayes optimal decision for the task *if* these were the true task distributions.

From a Bayesian point of view, the search for the optimal model can be seen as optimizing a posterior on the model space:

$$\Pr(\mathrm{M}|\mathbf{z}) \propto \Pr(\mathbf{z}|\mathrm{M})\Pr(\mathrm{M})$$

where M represents a model in $\mathcal{M}$ and $\mathbf{z}$ is the training data.

Since this is a classification task, the data likelihood, $\Pr(\mathbf{z}|\mathrm{M})$ should be based on the empirical classification loss and *not* the data modeling probability within the individual class. For example, $\Pr(\mathbf{z}|\mathrm{M}) = \prod_i \Pr(y_i|x_i, \mathrm{M})$ but not $\Pr(\mathbf{z}|\mathrm{M}) = \prod_i \Pr(x_i, y_i|\mathrm{M})$. In our algorithm, we directly estimate $\Pr(\mathbf{z}|\mathrm{M})$ using the empirical error rate of a classifier trained using the augmented data set.

For $\Pr(\mathrm{M})$, it is natural to assign less probability to models that are more complex. Since, by design, we can only reach a more complex model by refining a current one (e.g. by refining the model for a specific object part), we can perform a best-first search on the model space based on the estimated $\Pr(\mathbf{z}|\mathrm{M})$ alone without the need to evaluate all possible models. Note also that this best-first search is consistent with the general search strategy that we outlined in Chapter 2.

3.3 Domain Theory

We illustrate our approach with the task of distinguishing similar Chinese characters. This section describes the prior domain knowledge about Chinese characters that we use in our experiments. We employ a rather simple domain theory for describing handwritten Chinese characters. Nevertheless, we will show that this already results in improved classification performance through the use of phantom examples.

Our domain theory consists of three parts. First, we provide a prototype model for each character. These prototypes can either be manually constructed by an expert, or extracted from an existing vector font database. Each prototype consists of several parts, where each part corresponds to a particular stroke in the character.

The second part of the domain theory consists of different models for each stroke. Here, we employ only two possible models: either a straight line segment or a quadratic Bezier curve segment. Using the notation from the previous section, the set of possible stroke models is $\Gamma^* = \{\Gamma_{line}, \Gamma_{curve}\}$. A character C_0 that consists of k strokes can therefore have 2^k different models. Both models assume a uniform width (i.e. stroke thickness) for each stroke. The line model uses 5 parameters for each stroke (the two endpoints, plus the width), while the curve model uses 7 (the 3 control points, plus the width). This can be summarized in our notations, with

$$Z_{\Gamma_{line}} = (x_1, y_1, x_2, y_2, w),$$

$$Z_{\Gamma_{curve}} = (x_0, y_0, x_1, y_1, x_2, y_2, w),$$

where $k(\Gamma_{line}) = 5$, and $k(\Gamma_{curve}) = 7$. In our actual system, a variation of the above parameterization is often used, where, instead of absolute coordinates, the parameters for each stroke Z_{Γ_i} (except the first stroke) are relative to the previous stroke:

$$Z'_{\Gamma_{line}} = (\delta x, \delta y, \delta\theta, \delta l, \delta w),$$

$$Z'_{\Gamma_{curve}} = (\delta x, \delta y, \delta\theta_1, \delta l_1, \delta\theta_2, \delta l_2, \delta w).$$

The re-parameterization is intended to reduce the dependency among the parameters, and to simplify the distribution (e.g. making it more Gaussian-like). This is illustrated in Figure 3.1.

The final part of our domain theory consists of a set of qualitative constraints between strokes in each character. These constraints are optional, but it allows the expert to incorporate additional information into the models. We use the 24 atomic relations between oriented line segments as introduced in [Moratz et al., 2000] for qualitative spatial reasoning. These relationships naturally extend to curve segments as well, and can be used to indicate and/or enforce intersections, junctions etc between strokes.

To specify any of the atomic relations between two *directed* line segments, four relationships must be given. Let p_1 and p_2 be the two endpoints of line segment L_p, q_1 and q_2 be the two end-

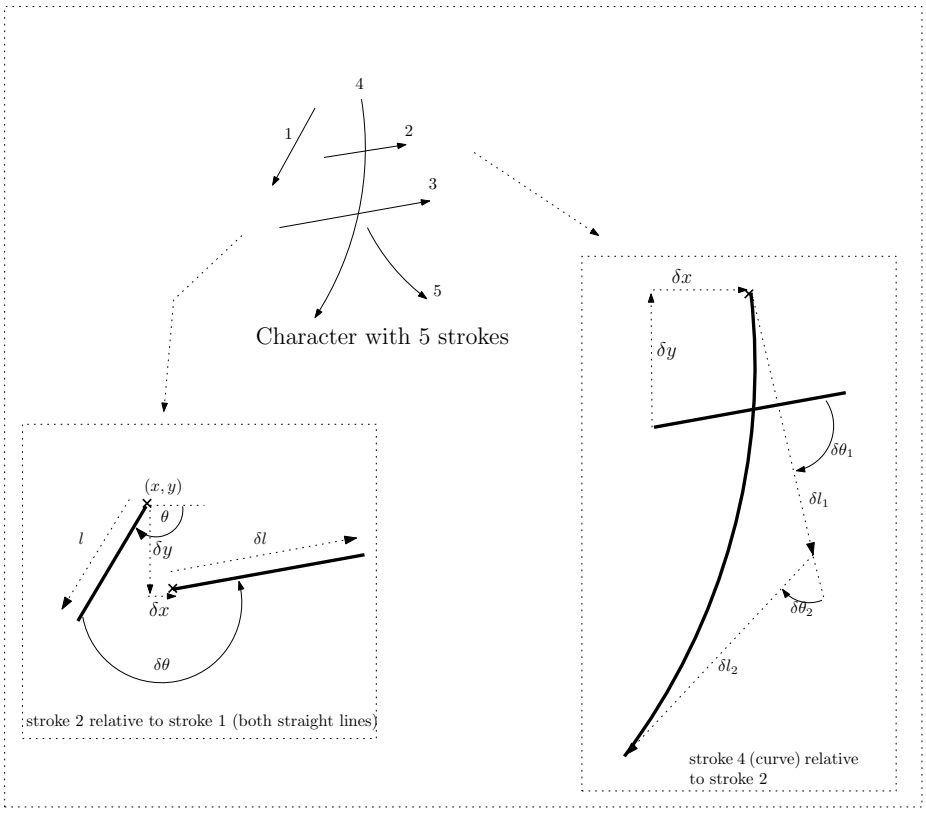


Figure 3.1: Relative parameters between strokes (line widths are not shown)

points of line segment L_q. The four relationships are:

- Location of p_1 relative to L_q
- Location of p_2 relative to L_q
- Location of q_1 relative to L_p
- Location of q_2 relative to L_p

Each relative location can be either left(l), right(r), start(s) or end(e). Figure 3.2 shows two example relationships.

3.4 Explaining the Examples

The process of explaining the examples is central in any EBL approaches. Functionally, it serves to reveal any hidden structure within each training example as predicted by the domain theory. From an information point of view, it is a way to exploit information within the input $\mathcal{X}$ itself.

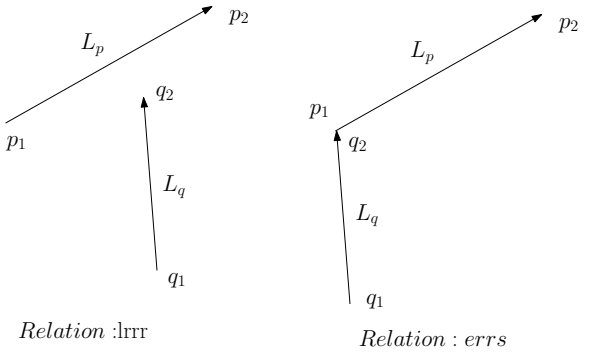


Figure 3.2: Two examples of qualitative constraints

"Classical" EBL requires that an explanation logically derives the class label from the observable inputs, using only statements contained within or derivable from the domain theory. Whenever a domain theory is adequate for the task, there exists a non-empty set of possible *inference paths* from any legitimate inputs to the class label. The actual training set serves to identify the subset of inference paths that are relevant to the particular task and to construct a hypothesis that is a generalization of this subset (see Fig. 3.3). Statistical robustness can be added by subjecting the derived hypothesis to test data, as proposed in [DeJong, 2006].

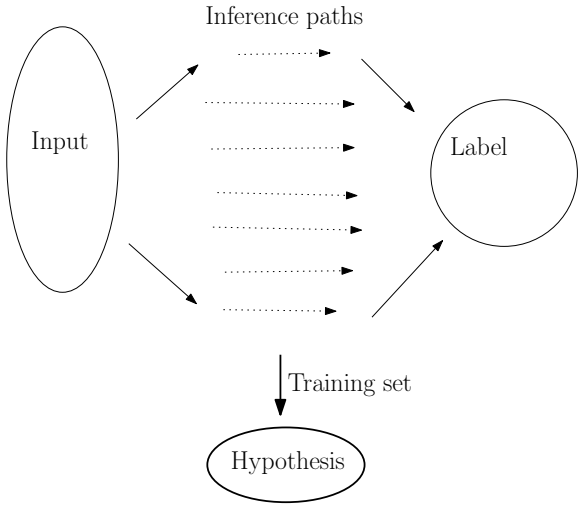


Figure 3.3: EBL

We note that in the classical EBL approach, the burden of being able to derive the label from the input lies entirely within the domain theory. This can be troublesome if the observable inputs are conceptually low-level features, such as pixels in image.

Instead of insisting on strictly logical inference, we can allow probabilistic inference by replac-

ing the set of possible inference paths with the set of possible generative models. The search of possible inference paths then becomes the search of useful models.

As described in the previous section, our model space is structured by decomposing the class object into parts. These parts are described by hidden variables Z. Explaining the training examples then involves finding the most likely Z that corresponds to each input X.

Fixing the model (i.e. Γ), the explanation process can be viewed as an instance of Bayesian inference, where a prior over possible Z_Γ is given by the domain theory. The posterior $\Pr(Z|X)$ is given by the Bayes rule:

$$\Pr(Z_\Gamma|X) = \frac{\Pr(X|Z_\Gamma)\Pr(Z_\Gamma)}{\Pr(X)}$$

Taking the logarithm, we find the Z_Γ that maximizes

$$\ln \Pr(Z_\Gamma|X) \propto \ln \Pr(X|Z_\Gamma) + \ln \Pr(Z_\Gamma)$$

If we define loss functions $\mathcal{H}(\phi(z), x) = -\frac{1}{\eta_1} \ln \Pr(x|z)$ and $\mathcal{L}(z) = -\frac{1}{\eta_2} \ln \Pr(z)$, (both η_1 and η_2 positive constants), then finding the most likely Z_Γ (denoted z^*) is equivalent to the following optimization problem:

$$\text{Find:} \qquad z^* = \arg\min_{z \in \mathcal{Z}} \mathcal{H}(\phi(z), x) + \eta \mathcal{L}(z) \tag{3.2}$$

where the constant $\eta = \frac{\eta_2}{\eta_1}$ is a weighting parameter balancing between $\mathcal{H}$ and $\mathcal{L}$.

The function ϕ is as defined in section 3.2.2. $\mathcal{H}$ is a distance metric between the two images x and $\phi(z)$, For $\mathcal{H}$, we used a Hausdorff-like distance [Rucklidge, 1996],

$$\mathcal{H}(x_1, x_2) = x_1 \cdot \mathcal{D}(x_2) + x_2 \cdot \mathcal{D}(x_1)$$

where $\mathcal{D}(x)$ represents the distance transform of image x (that is, $\mathcal{D}$ gives the distance from each pixel to the nearest nonzero pixel). $\mathcal{L}$ penalizes violation of any qualitative constraints (see section 3.3) specified by the prototype character. We penalize each violation by the minimal distance that the target point has to move in order to satisfy the qualitative relationship.

The final, tuned generative model is a distribution over Z. Notice that in the above setting we have a prior $\Pr(Z)$ based on the domain theory and we use this to find the most likely Z for individual example. The Z's for individual example are then used to estimate the generative model, parameterized by θ. If we treat Z as latent variables, we can in fact find the generative model without the need to find the individual Z's using an EM-like algorithm. We do not pursue this

due to the complexity involved, and to avoid potential overfitting. Instead, we simply trust the prior as given indirectly by the domain theory.

3.5 Algorithm

Given a binary classification task between character C_0 and C_1, our algorithm attempts to construct an adequate model for this task by performing a greedy search in the structured space $\mathcal{M}^{C_0,C_1}$ of possible models. The "search operator" here is a refinement operation, where a part of the current model is replaced with a more complex one. For simplicity, we assume that a model with more parameters is more complex than one with less parameters. Therefore, the curve model Γ_{curve} is considered more complex than the line model Γ_{line}. It follows that, the simplest model in $\mathcal{M}^{C_0,C_1}$ is $\langle \Gamma^{C_0}, \Gamma^{C_1} \rangle$ with $\Gamma^{C_0} = \langle \Gamma_{line}, \ldots, \Gamma_{line} \rangle$ and $\Gamma^{C_1} = \langle \Gamma_{line}, \ldots, \Gamma_{line} \rangle$.

The algorithm is outlined below:

1. Pick the simplest model $M = \langle \Gamma^{C_0}, \Gamma^{C_1} \rangle$ from $\mathcal{M}^{C_0,C_1}$ to be used as the initial model.
2. Repeat
   (a) Estimate the parameters Z of the current model M for each training example and learn a distribution over Z.
   (b) Generate phantom examples from the calibrated model.
   (c) Learn a classifier with the augmented training set and perform cross-validation to estimate its classification accuracy.
   (d) If the accuracy is satisfactory, or if no further refinement to M can be made, stop.
   (e) Pick a single stroke component Γ_i from vector $\langle \Gamma^{C_0}, \Gamma^{C_1} \rangle$ such that its refinement would result in a model M' that best fits the training character images. We then replace the current model M with the more sophisticated M'.

Step 2a requires that we extract stroke-level parameters from each training image. This would be easy if online sample is used since the stroke information is readily available. For offline sample where only pixel information is available, we perform the following optimization as described in Section 3.4. We use a simulated-annealing algorithm to perform the optimization and in most cases the automatically extracted strokes are satisfactory (close to what a manual stroke-labeling would have produced).

We model the distribution for Z as a multivariate Gaussian, where the mean and the covariance are estimated from the extracted stroke parameters in the above step.

For step 2b, each phantom example is generated by first sampling from the calibrated Gaussian distribution, then tested against the qualitative constraints. Examples violating the constraints are simply discarded. This rejection sampling process results in a rather complicated generative model which reflect our knowledge about Chinese characters.

For step 2c, we perform 5-fold cross-validation on an augmented training set using a support vector machine with a Gaussian kernel. 5000 phantom examples are drawn from the calibrated model for each fold of the cross-validation. The cross-validation is repeated five times (i.e. a total of 25 SVM trainings are performed) and the average accuracy is recorded.

If the classification accuracy is not satisfactory, and if there are still possible refinements, i.e., there are still strokes that use the simpler stroke model Γ_{line}, we pick a stroke for refinement by replacing it with the more sophisticated model Γ_{curve} in step 2e. Each refined model is evaluated by locally refitting the new stroke parameters for each training image using the same objective function as in Eq.(3.2). The one that achieves the best fit (i.e. lowest total loss) among the candidate refinements is chosen.

By "satisfactory" we mean that the cross-validation error rate is lower then a pre-specified threshold. A low threshold will ensure that the system works "harder" to find a good model, and vice versa.

3.6 Experiments

Distinguishing offline, handwritten Chinese characters images is a challenging learning task. The greater complexity of Chinese characters necessitates higher resolution images. We use ETL9B which uses 63×64 pixels for each image, compared to 20×20 in the NIST digit databases. With thousands of common Chinese characters no database begins to rival those of digits or Latin characters. ETL9B contains only 200 labeled handwritten images for each class.

We illustrate our approach by first focusing on the four characters shown in Figure 3.4 which form six pair-wise classification tasks of different difficulties. Pair 1-2 is particularly hard; 1-3 and 2-3 are moderate. Character 4 is very different; its three classification tasks are all easy.

Experiments 1 to 8 are in-depth evaluation of the proposed approach based on these pairs. In Experiment 9, we apply our algorithm to other, more challenging pairs of Chinese characters.

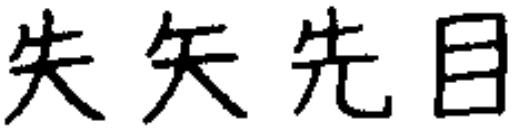


Figure 3.4: The characters 1,2,3,4 (left to right)

For the construction of a complete classification system for Chinese characters, we propose the use of a two-stage classifier, as in [Prevost et al., 2005]. The first stage classifier can be a fast, "coarse-grained" classifier that can quickly and reliably identify a small set of possible candidate labels. The second stage classifier would then focus on distinguishing mutually confusing characters within such sets. The second stage is where more domain knowledge and modeling effort is needed, and our approach can be applied.

We use soft-margin SVM with RBF kernels as our discriminative learner. The input to the SVM consists of vectorized images, each is a 4032-dimension vector of pixel values. SVM parameters are chosen using grid-search and cross-validation over only the real training set. All our presented results are based on 5-fold cross-validation. Unless otherwise stated, experiments are repeated 10 times and error bars denote the 95 percent confidence interval.

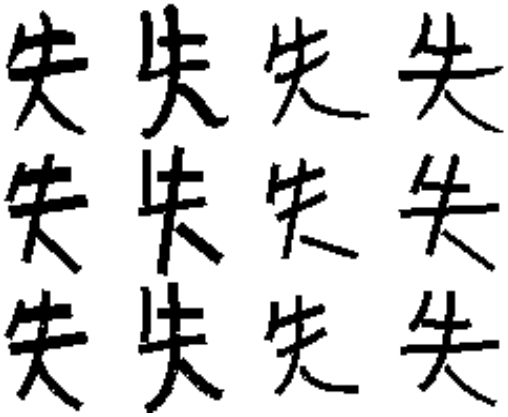


Figure 3.5: Examples of character 1: Real (top row), phantom with straight lines (middle row), phantom with curves (bottom row)

The actual phantom examples are pixel images derived from the stroke descriptions. Figure 3.5 shows four real training examples and their corresponding approximations with the simple all-straight-line as well as the complex all-curve phantom generative models. Even with the most complex models according to our domain theory, the phantom examples still have noticeable dif-

ferences (sharp corners, uniform width etc) from the real ones, since our domain theory is only approximate.

Our experiments did not treat phantom examples differently than real examples. Phantom and real examples are provided undifferentiated to the SVM. Properly weighting phantom examples less could potentially improve the overall performance.

3.6.1 Experiment 1: Discriminative Information from Domain Knowledge

Our first set of experiments explores the nature of the discriminative information resulting from our simplest models where all strokes are modeled as straight lines. In Experiment 1.1, we varied the number of phantom examples (M) from zero to 10,000. Zero phantom examples is equivalent to conventionally training the native SVM on the real training examples only. As the phantom training set is increased, the information (good or bad) from the constructed generative models is more completely forced into the learner. We reason that if the generative models provide useful information, the error rate should decrease with additional phantom training. If not, more phantom examples will increase the error rate over the base-line level. The results are shown in Figure 3.6

The error rates in the difficult and medium-difficult learning problems decrease significantly with additional phantom examples. On the easy pairs, performance is already at ceiling and phantom training has little discernable effect. In no condition and at no level of training do phantom examples significantly harm performance.

3.6.2 Experiment 2: Dynamic Model Construction

In this experiment we demonstrate our system's ability to adaptively construct appropriate character models based on the classification problem at hand and the desired error rate specified using the algorithm proposed in section 3.5. The error rates are obtained by averaging five 5-fold cross-validations, with the training set augmented with 5000 phantom examples generated using the character models.

We have identified the relative difficulties to classify the six pairs of characters in the previous experiment. When we test our algorithm with the most difficult pair, with an error threshold of 8% or higher, the algorithm selects the simplest character model consisting of only straight-line stroke models, which achieves a 7.05% error rate, as illustrated in Figure 3.7a. If we require an er-

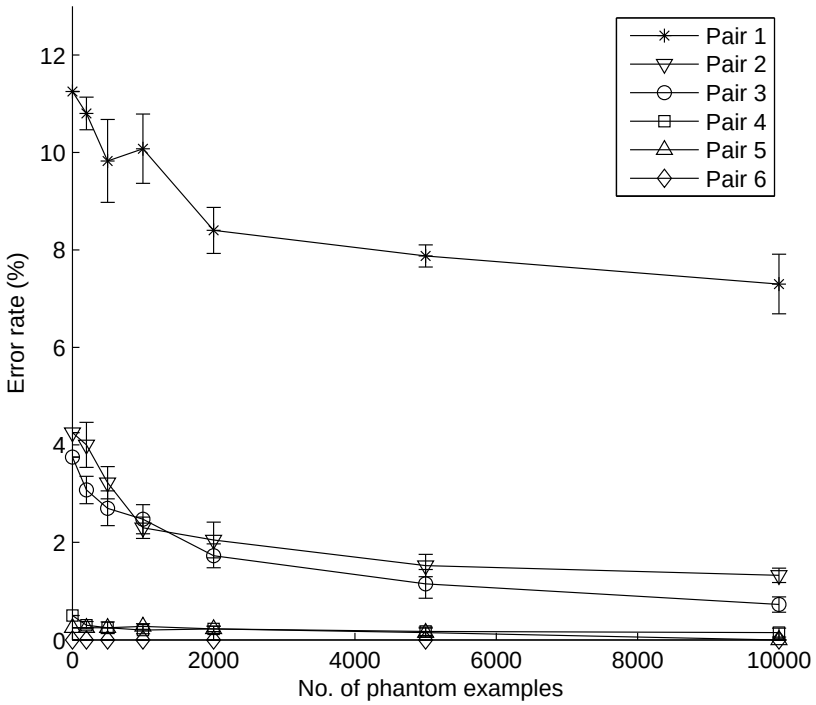
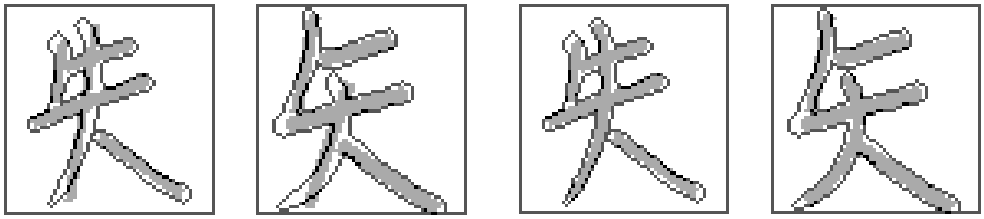


Figure 3.6: The effect of varying number of phantom examples

ror rate of 6% or better, the algorithm refines the longest, most curved stroke in each character to a curve model, and achieves an 5.1% error rate (see Figure 3.7b). Refining the character models further only gives marginal improvements or even hurts the classification accuracy, as can be seen in Figure 3.8.

If we apply our algorithm to one of the easiest pairs, with the error rate thresholds of 8%, 6%, or even 1%, the algorithm will still choose the simplest all straight-line model for both characters, as shown in Figure 3.9a. It is only when we require a 0.1% error threshold that the algorithm chooses to refine the most curved stroke in the first character, as in Figure 3.9b.

By using phantom examples and automated model construction, we managed to obtain performance close to that achieved with state-of-the-art, hand-tailored features for handwritten Chinese characters, for example, the weighted directional code histogram [Kimura, 1997].



(a) (b)

Figure 3.7: 3.7a shows the character models chosen by the algorithm for the hardest pair when an error rate threshold of 8% or higher is specified. The contours represent the real character images, and the shaded areas are the representations of our character model. Straight line models are used in all strokes in both characters. 3.7b shows the character models chosen by the algorithm when an error rate threshold of 6% is specified. Straight line models are used in all strokes except the longest, most curved stroke in both characters.

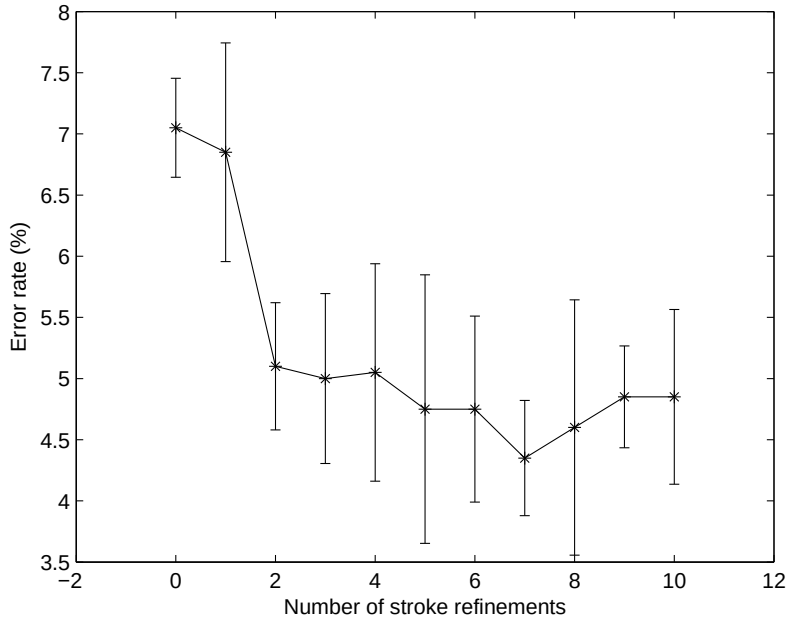
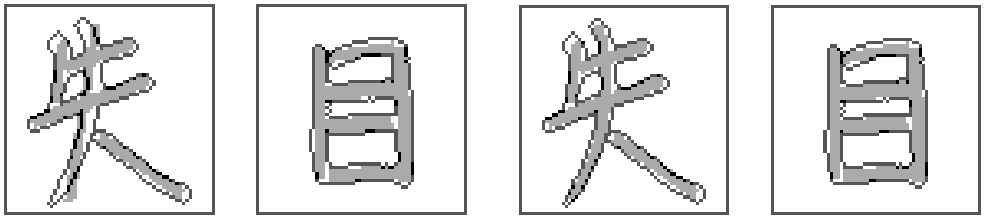


Figure 3.8: Averaged 5-fold cross-validation error rates given different number of model refinements (i.e. at different stages of model construction).



(a) (b)

Figure 3.9: 3.9a shows the character models chosen by the algorithm for an easy pair when an error rate threshold of 8%, 6%, or 1% is specified. The contours represent the real character images, and the shaded areas are the representations of our character model. Straight line models are used in all strokes in both characters. 3.9b shows the character models chosen by the algorithm when an error rate threshold of 0.1% is specified. Straight line models are used in all strokes except the longest, most curved stroke in the first character.

3.6.3 Experiment 3: Effect of Varying Number of Real Examples Used in Phantom Training

In this experiment we further investigate properties of our generative models. We believe that the quality of the constructed generative models improves uniformly as more real examples are used in their construction. To test this prediction we vary the size of the real training set ($\tilde{N}$) from less than 10 per class to 320 (160 per class) on the hardest pair (1-2). The results are shown in Figure 3.10. The first condition (dotted line) is the native SVM. The second condition (solid line) is the same SVM but with an additional 5000 phantom examples sampled from the generative models constructed from the very same real training examples. We include results for phantoms trained with the simplest model (all straight lines) as well as those with a more refined model (the longest strokes modeled with curves).

The results show that the generative models always help this hardest classification problem. Similar results seem to hold for the medium-hard problems. Interestingly, even relatively small real training sets can produce beneficial phantom examples.

Figure 3.11 shows the same experiment results in different form. Here we measure the improvement afforded by the generative models. We plot the (approximate) number of additional real training examples that the native SVM would have required to match the error rate of the combined real+phantom training set. The benefit is always positive and we observe that the phantom examples trained with a refined model "worth" more than those with a simple model.

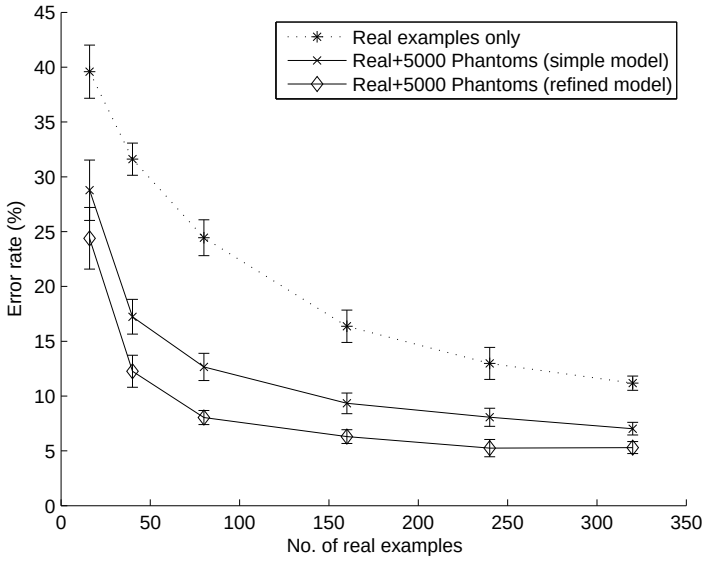


Figure 3.10: Performance gain with phantom examples trained with varying number of real examples

3.6.4 Experiment 4: Interaction is Crucial

When $\tilde{N}$ (i.e. the actual number of real examples used to tune the phantom generative model) is small (or zero), the generative model is only tuned with very little "real" information. We claim that this lack of "interaction" between the domain knowledge and the real training examples will not benefit (and may even harm) the performance.

We test this hypothesis with Experiment 4. For the non-interaction case, we employ a prior generative model to generate the phantom examples as before. But this prior model is not tuned with real examples, thus it allows a larger variance in the stroke parameter space, although each is still subjected to the same set of qualitative constraints that are used in the tuned case (i.e. those violating the constraints are still discarded). Experiment 4 compares two conditions: interaction (the solid lines) and non-interaction (the dotted lines) on two learning pairs: the hardest pair 1-2 (top) and a medium hard pair 1-3 (bottom).

Figure 3.12 shows the results. For the medium-hard pair, the non-interaction phantom examples do not appear to help. For the hard pair, they significantly harm performance. The result clearly rejects the alternative hypothesis; merely the additional information of domain theory without allowing explanation and calibration interaction with training examples cannot explain the observed benefit.

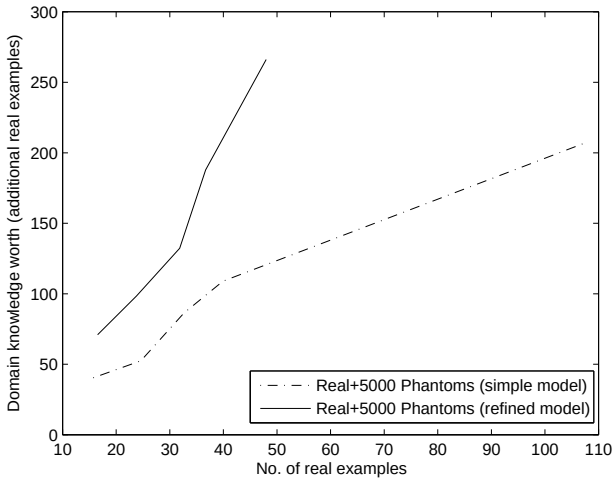


Figure 3.11: Domain knowledge measured in terms of additional real examples

3.6.5 Experiment 5: Domain Knowledge Helps More with Fewer Real Training Examples

Our analysis claims that when sufficient number of real examples are used, imperfect domain knowledge will eventually affect the performance negatively. We test this hypothesis with Experiment 5.

This setup of this experiment is similar to Experiment 3, except that we purposely limit the number of real examples used to tune the generative model, in order to obtain an artificially impoverished domain theory.

Figure 3.13 shows the results when 16 and 40 real examples are used to tune the generative models. 5000 phantom examples are generated and added to the training set in each case. Note that the x-axis shows the number of real examples used in the training set (for the SVM – not for tuning the generative model). When fewer real examples are used to tune the model, the contribution of phantom examples worth less. When only 16 real examples are used to tune, beyond about 200 real training examples, the performance is actually worse when phantom examples are used.

3.6.6 Experiment 6: Discriminative Classifier Helps

We incorporate domain knowledge into the learning problem through the use of generative models estimated from the real training examples. Since these generative models are suggested by the

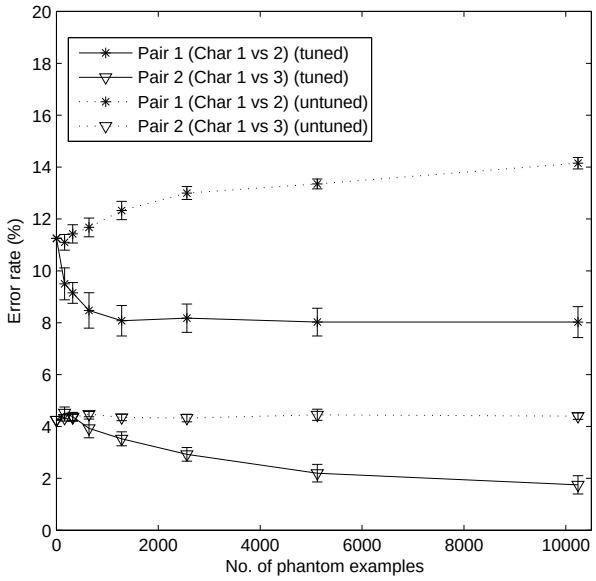


Figure 3.12: Effect of interactions

domain theory, one would then question whether the trained generative models are already good enough for the classification task.

Pair	Generative (% error)	Discriminative (% error)
1	38	11.25
2	23.75	4.25
3	18.25	3.75
4	1.5	0.5
5	0.75	0.25
6	1.5	0

Table 3.1: Generative vs Discriminative Classifier

To answer this question, we use the tuned generative models to classify test examples based on likelihood of the stroke parameters of each example with respect to each character class. Table 3.1 shows the results. The generative classifiers perform much worse, and the gap is larger for harder tasks. This verifies that our generative models are indeed only (very) approximate, and the use of discriminative learner is essential in order to benefit from our generative models.

3.6.7 Experiment 7: Robustness in the Presence of Noise

A crucial issue of machines learning is robustness in the presence of noise. Here, we focus on noise in the labels (i.e. part of the training set is wrongly labeled). When an example is wrongly labeled, the "explanation" process should indicate that our confidence on that particular example is

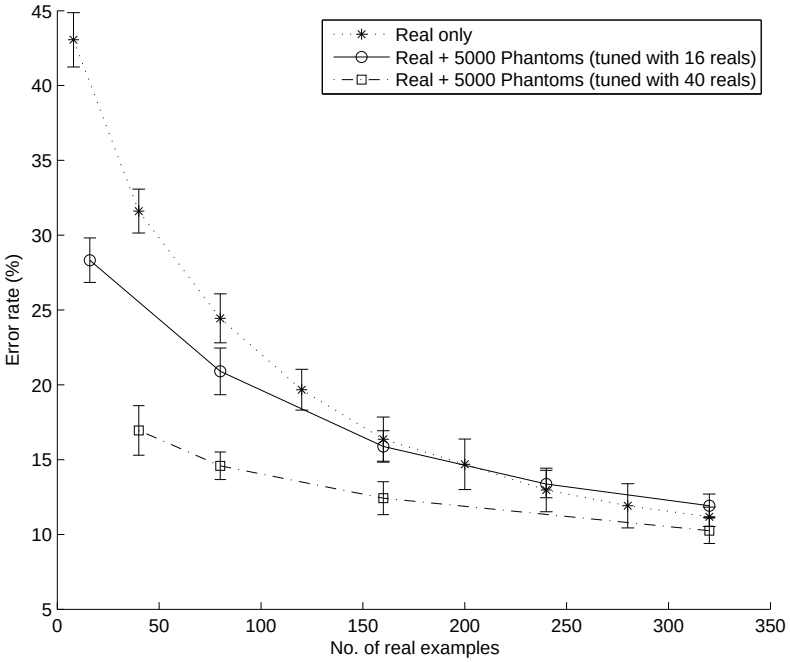


Figure 3.13: Performance gain with phantom examples trained with fixed number of real examples

lower than in cases when the labels are correct. In the case of tuning phantom examples, the estimated parameters of the generative model may be affected by such label noise, but the overall structure of the model would still respect the inherent stroke structure given by the domain theory. The use of phantom examples would then serve to bias the learner such that it is less affected by such noise, and therefore achieve better robustness.

We test this by injecting varying amount of noise in the labels. In each instance of the experiment, the labels of some randomly chosen subset of the real examples are flipped. Each training set is augmented with phantom examples (that are correctly labeled). We evaluate both the "perceived" error rate and the "true" error rate (i.e. if correct labels are used) during training and testing.

Figure 3.14 shows these error rates without using phantom examples, on the hardest pair. In terms of the training error, the true error is higher than the perceived error. This indicates some amount of overfitting to the training set. However, in terms of the test error, the true error is consistently less than the perceived error. This exposes a major strength of the soft-margin SVM in terms of robustness to noise.

We predict that with phantom examples, an even higher level of robustness is achievable due to strong bias from domain knowledge. Figure 3.15 shows the results when each training set is

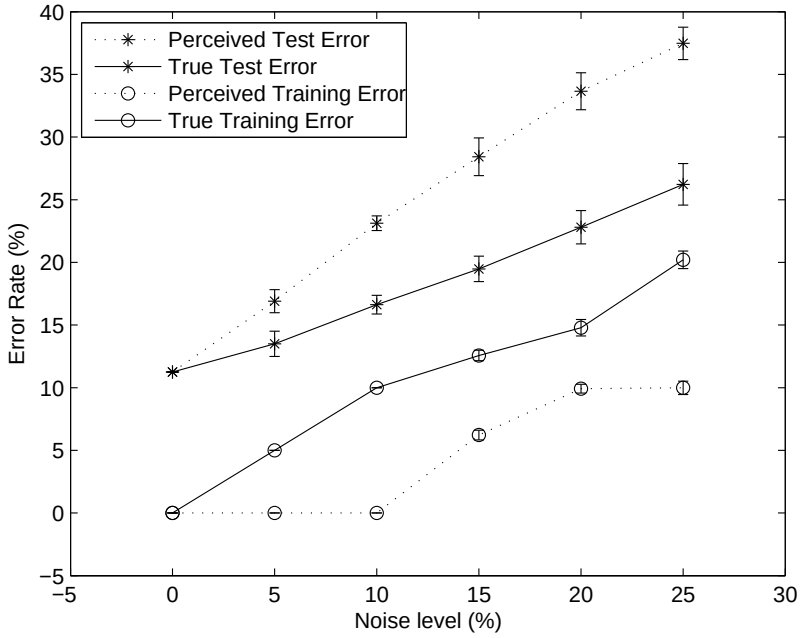


Figure 3.14: Effect of label noise (without phantoms)

augmented with 2500 phantom examples. We see that the augmented training sets manage to improve the performance on the test set (both true and perceived errors) even when the phantom examples are tuned using noisy examples. Moreover, there is a larger gap between the perceived and the true test error, indicating a higher level of robustness to noise. When the noise level is high, we observe a remarkable phenomenon where the true test error is actually lower than the perceived training error. This indicates a strong bias from the domain knowledge against overfitting to wrongly labeled examples.

3.6.8 Experiment 8: Comparison with Virtual Support Vectors

This final experiment compares phantom examples with virtual support vectors [Burges and Schölkopf, 1996; Decoste and Schölkopf, 2002] in terms of improvement in classification error rate. The virtual support vector method creates new virtual examples from the real ones by applying known invariance transformations (i.e. transformations that do not alter the label of the examples) to the real examples. [Decoste and Schölkopf, 2002] tested translations of each training example by 1 pixel in all 8 directions, and 2 pixels in the horizontal and vertical directions (a total of 12 different translations). It was also reported that rotations and varying stroke thickness help. We

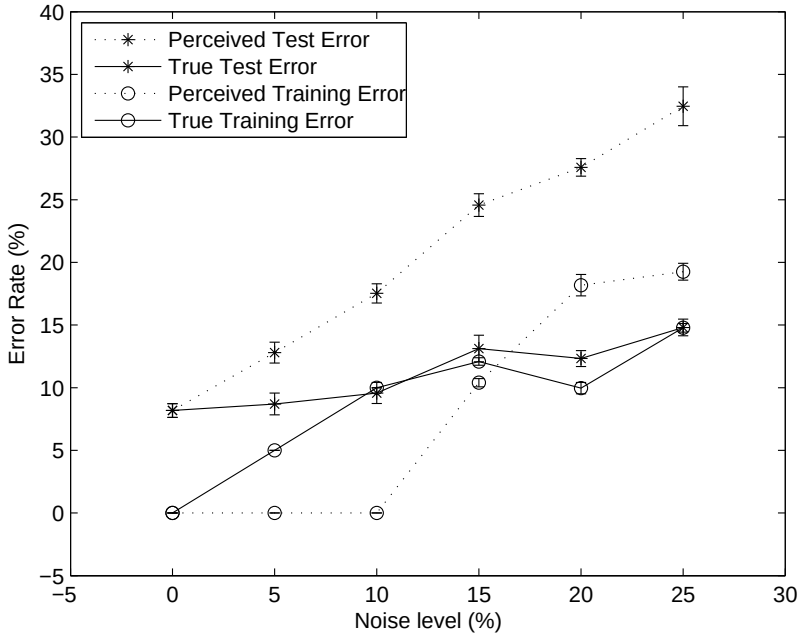


Figure 3.15: Effect of label noise (with phantoms)

employ the same transformations and evaluate the performance on the hardest pair. Figure 3.16 shows the results.

The horizontal lines depict the error rates achieved by the virtual sample method. For the combination of translations, rotations and thickness, we reduce the number of translations (and replace them with varying thickness) due to memory limit. Combining both translations and rotations does improve the performance, while replacing some translations with thickness variations does not help (the two horizontal lines overlap – the error rate is the same as using just the 12 translations).

From our point of view, the invariants are just another form of domain knowledge, easier to apply but weaker in the sense that it is pre-determined and does not allow much interaction with the training examples. The experiment results show that better error rates can be achieved with phantom examples, and there is still room for further improvement.

3.6.9 Experiment 9: More Challenging Pairs

For this experiment, we first identify the most difficult pairs of Chinese characters from the ETL9B database. We first learn a standard multiclass classifier (with multiclass linear discriminant) for

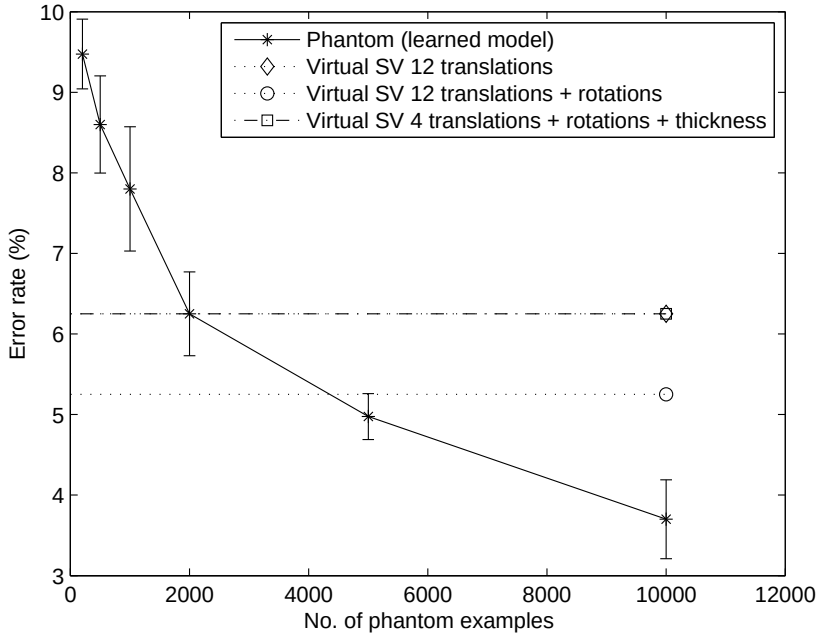


Figure 3.16: Comparison with virtual support vectors

the entire set of more than 3000 Chinese characters in the database. Next, for each error, the confusing pair of characters is noted. The top 10 most easily confused pairs of characters are then identified. They are shown in Figure 3.17.

Although a human expert can easily tell them apart, each pair in this set is different in only minor details that may not be easily picked up by a machine learner. We evaluate our approach on these 10 pairs, and as in experiment 8, compare the results with that of the virtual support vectors (12-neighbor translations and up to 6 degree of rotations). Figure 3.18 shows the results where 5000 artificial examples are used in each trial.

Significant performance improvements are achieved in all 10 tasks. In all but one of the pairs, better error rates are achieved with the phantom examples (with the same number of real and artificial examples).

We believe that the two types of artificial examples (those from the virtual SV approach and those from the phantom approach), provide very different kinds of information to the discriminative learner. The virtual support vectors are transformed versions of the real examples and still look very similar to the real ones. On the other hand, the phantom examples are manufactured from the learned generative models and can look very different from the real ones (shape corners,

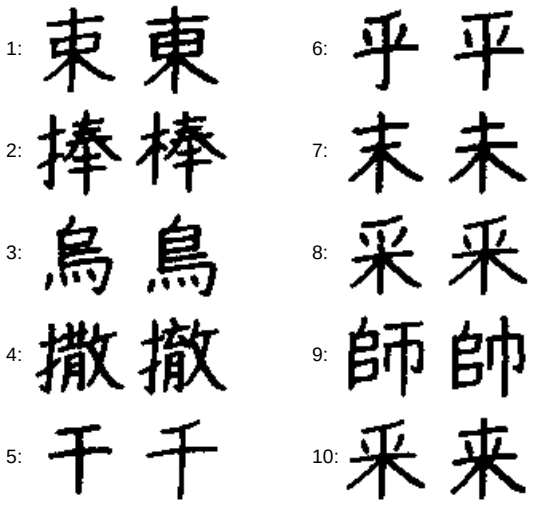


Figure 3.17: Some of the most challenging pairs of characters in ETL9B

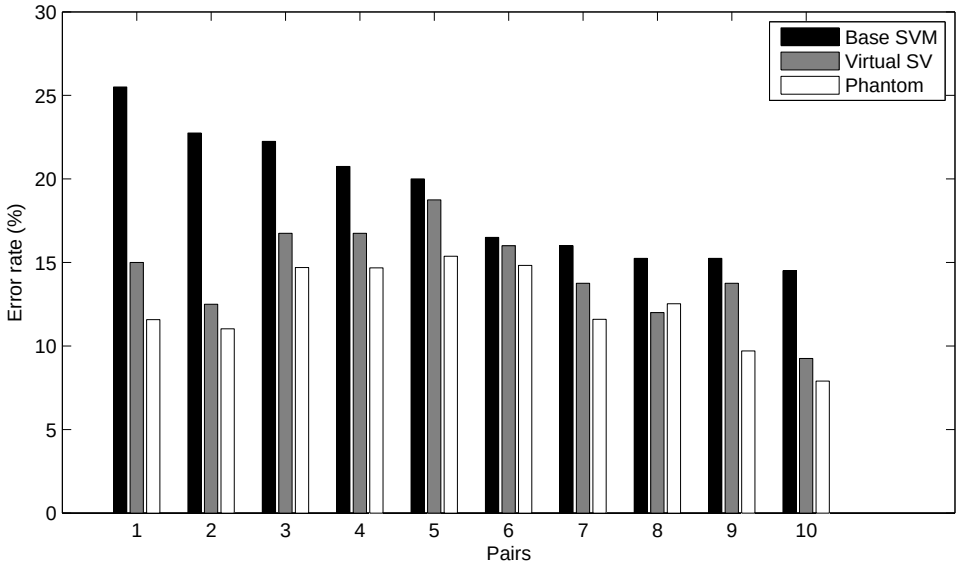


Figure 3.18: Errors for the challenging pairs

uniform width etc). The information content of both types of examples, although may not be entirely independent, can complement each other and an adaptive, hybrid approach may potentially achieve better performance from the same number of real and artificial examples.

Chapter 4

Simple Discriminative Semantic Features

4.1 Introduction

Sensitivity to appropriate features is crucial to the success of supervised learning. The appropriateness of a feature set depends on its relevance to the particular task. Various notions of relevance have been proposed [Blum and Langley, 1997; Kohavi and John, 1997] and statistical tools to evaluate and select relevant feature sets are available (e.g. hypothesis testing, random probes, cross-validation). Unfortunately, most feature construction strategies focus on feature subset selection (e.g. filters [Almuallim and Ditterich, 1991; Kira and Rendell, 1992], wrappers [Kohavi and John, 1997], embedded methods [Guyon et al., 2006]) from either a pre-determined set or a space defined by some feature construction operators [Markovitch and Rosenstein, 2002] and require many training examples to evaluate.

In challenging domains, the "native", observable features (e.g. pixel values of an image) are high-dimensional and encode a large amount of mostly irrelevant information. There are standard statistical techniques such as principal components analysis (PCA) and linear discriminant analysis (LDA) to reduce the dimensionality of the input features. But often the features that simplify the task are complex nonlinear combinations of the native features. Increasing the complexity of the hypothesis space to include nonlinear combinations will require an impractical amount of training data. We believe that the most effective approach is to incorporate additional information in the form of prior domain knowledge. In this direction, one approach [Gabrilovich and Markovitch, 2005] utilizes the large repository of the Open Directory Project to aid in natural language classification and the proposed feature generation technique results in significantly improved performance.

Most feature generation approaches, however, still rely on first augmenting the current feature set with potentially useful features, followed by a feature selection process. The relevance of features largely depends on their empirical predictive performance on the labels. We would like to

utilize prior domain knowledge in determining the relevance by incorporating information in the form of "analytical evidence", in addition to the empirical performance. Explanation-based learning (EBL) is a method of dynamically incorporating prior domain knowledge into the learning process by explaining training examples. The explanation process serves as the source of analytical evidence. In character recognition tasks for example, we know that not all pixels in the input image are equally important. [Sun and DeJong, 2005] used this observation to learn specialized Feature Kernel Functions for support vector machines. These embody a specially constructed distance metric that is automatically tailored to the learning task at hand. That approach automatically discovers the pixels in the images that are more helpful in distinguishing between classes; the feature kernel function magnifies their contribution.

However, it is not the raw pixels that intrinsically distinguish one character from one another. Rather, the pixels are due to strokes of a writing implement and these strokes in relation to one another distinguish the characters. Not telling the learner of these abstract relationships means that it must perform the moral equivalent of inventing the notion of strokes from repeated exposure to patterns of raw pixels, making the learning task artificially difficult. We would like the feature construction process to appreciate such abstractions of the domain theory and perform a "dynamic" feature selection. This allows, for example, the use of different pixel subset or distance metric for individual character image.

4.2 Semantic Features and Generalization

The criteria for feature construction are usually based on how well a particular set of features retains the information about the class while discarding as much irrelevant and redundant information as possible. Information-theoretic methods, based on mutual information, channel coding and rate-distortion theorems have been applied to this problem [Battiti, 1994; Tishby et al., 1999; Torkkola, 2003]. Let X and Y be random variables that represent the input and the label respectively, and there is an underlying joint-distribution on (X, Y). It can be shown [Akaike and Merhav, 1994] that the optimal Bayes error is upper bounded by $\frac{1}{2}H(Y|X)$, where H is the entropy in bits. Therefore, a feature $F(X)$ that retains as much information as possible about the label will have $H(Y|F(X)) \leq H(Y|X) + \epsilon$, where $\epsilon > 0$ represents the amount of information loss that we are willing to tolerate. On the other hand, discarding irrelevant information can be achieved by minimizing $H(F(X)|Y)$ while satisfying the condition on $H(Y|F(X))$. Intuitively, this implies

that most information about the class label is preserved, while irrelevant information (e.g. within-class variations) is discarded.

We show how a feature $F(X)$ with small $H(F(X)|Y)$ and small $H(Y|F(X))$ leads to better generalization bound in terms of Rademacher complexity, for the specific case of binary classification. We make use of the following theorem:

Theorem 1. *[Bartlett and Mendelson, 2002] Let D be a probability distribution on $\mathcal{X} \times \{\pm 1\}$, let G be a set of $\{\pm 1\}$-valued functions defined on $\mathcal{X}$, and let $(X_i, Y_i)_{i=1}^m$ be training samples drawn according to D^m. With probability at least $1-\delta$, every function g in G satisfies*

$$Pr_D(Y \neq g(X)) \leq \hat{P}_m(Y \neq g(X)) + \frac{R_m(G)}{2} + \sqrt{\frac{ln(1/\delta)}{2m}}$$

where $\hat{P}_m$ is the empirical risk. $R_m(G)$ is the Rademacher complexity of G, given by

$$R_m(G) = \mathbf{E}_{\sigma_{1\ldots m}} \mathbf{E}_{X_{1\ldots m}} \Big[\sup_{g \in G} |\frac{2}{m} \sum_{i=1}^{m} \sigma_i g(X_i)| \, \Big| X_1, \ldots, X_m \Big]$$

where $\sigma_1, \ldots, \sigma_m$ are independent $\{\pm 1\}$-valued random variables.

Given a feature F, where $F(X)$ takes values in $\mathcal{S}$. Let $\mathcal{G} : \mathcal{S} \rightarrow \{+1, -1\}$ be a space of functions. We make the following assumption:

Assumption 1. *Given $H(F(X)|Y) \leq H(F(X)) = \beta$, there exists a finite set $\mathcal{F}_t \subset \mathcal{S}$ of k different "typical" values (i.e. $|\mathcal{F}_t| = k$) such that $\Pr(F(x) \in \mathcal{F}_t) > 1 - \epsilon$ for some $\epsilon > 0$. It is straightforward to show that $k \approx 2^\beta$ when ϵ is small.*

The following lemma shows that $R_m(\mathcal{G})$ is bounded if $H(F(X)|Y)$ is bounded.

Lemma 1. *By assumption 1, for all $0 < \delta < 1$,*

$$R_m(\mathcal{G}) \leq 2(\delta + \epsilon) + 2\sqrt{\frac{2^\beta}{m}} + \sqrt{\frac{2}{m} \ln \frac{1}{\delta}}$$

Proof. Let $\mathbf{F} = (F(X_1), \ldots, F(X_m))$ be the sequence of feature values of the training sample $(X_1, \ldots, X_m)$. Let $\mathbf{I}_t(\mathbf{F}) = \{i | F(X_i) \in \mathcal{F}_t\}$ be the indices of examples with "typical" feature values, and $n_t(\mathbf{F}) = |\mathbf{I}_t(\mathbf{F})|$. Then, consider $R_m(\mathcal{G})$ with respect to the set of typical and non-typical

values,

$$\begin{aligned} R_m(\mathcal{G}) &= \mathbf{E}_{\mathbf{F}}\mathbf{E}_{\sigma_{i,\ldots,m}}\Big[\sup_{g\in\mathcal{G}}|\frac{2}{m}\sum_{i=1}^{m}\sigma_i g(F(X_i))|\Big|\mathbf{F}\Big] \\ &\leq \mathbf{E}_{\mathbf{F}}\Big[\mathbf{E}_{\sigma_i|i\in\mathbf{I}_t(\mathbf{F})}\big(\sup_{g\in\mathcal{G}}|\frac{2}{m}\sum_{i\in\mathbf{I}_t}\sigma_i g(F(X_i))|\big|\mathbf{F}\big) + \\ &\quad \mathbf{E}_{\sigma_i|i\notin\mathbf{I}_t(\mathbf{F})}\big(\sup_{g\in\mathcal{G}}|\frac{2}{m}\sum_{i\notin\mathbf{I}_t}\sigma_i g(F(X_i))|\big|\mathbf{F}\big)\Big] \\ &\leq \mathbf{E}_{\mathbf{F}}\Big[\mathbf{E}_{\sigma_i|i\in\mathbf{I}_t(\mathbf{F})}\big(\sup_{g\in\mathcal{G}}|\frac{2}{m}\sum_{i\in\mathbf{I}_t}\sigma_i g(F(X_i))|\big|\mathbf{F}\big) + \\ &\quad \frac{2}{m}(m-n_t(\mathbf{F}))\Big] \end{aligned}$$

For training examples with typical values, we group these into k groups, where the members within each group all have the same feature value. Regardless of function g, all members X_i within the same group must have the same $g(X_i)$. Let $\mathbf{I}_t^j(\mathbf{F})$ be the set of indices for members of group j and $n_t^j(\mathbf{F}) = |\mathbf{I}_t^j(\mathbf{F})|$, $j = 1,\ldots,k$. The expected absolute sum is therefore independent of $\mathcal{G}$, with

$$\begin{aligned} \mathbf{E}_{\sigma_i|i\in\mathbf{I}_t(\mathbf{F})}(|\frac{2}{m}\sum\sigma_i|) &\leq \frac{2}{m}\sum_{j=1}^{k}\mathbf{E}_{\sigma_i|i\in\mathbf{I}_t^j(\mathbf{F})}\big(|\sum_{i\in\mathbf{I}_t^j(\mathbf{F})}\sigma_i|\big) \\ &\leq \frac{2}{m}\sum_{j=1}^{k}\sqrt{n_t^j(\mathbf{F})} \\ &\leq \frac{2}{m}\sqrt{kn_t(\mathbf{F})} \end{aligned}$$

We therefore have,

$$R_m(\mathcal{G}) \leq \mathbf{E}_{\mathbf{F}}\Big[\frac{2}{m}\sqrt{kn_t(\mathbf{F})} + \frac{2}{m}(m-n_t(\mathbf{F}))\Big]$$

By assumption 1, the probability of obtaining a typical feature value is at least $1-\epsilon$ and therefore the distribution of $n_t(\mathbf{F})$ is binomial with mean $m(1-\epsilon)$. By Hoeffding's inequality,

$$\Pr(n_t(\mathbf{F}) \leq m(1-\epsilon)-\gamma) \leq e^{-\frac{2\gamma^2}{m}} \quad .$$

Let $\delta = e^{-\frac{2\gamma^2}{m}}$. Then, using $n_t(\mathbf{F}) = 0$ with probability δ and $n_t(\mathbf{F}) = m(1-\epsilon)-\gamma$ with probability

$1-\delta$, we have

$$\begin{aligned} R_m(\mathcal{G}) &\leq \delta(2) + (1-\delta)\frac{2}{m}\Big(\sqrt{k(m(1-\epsilon)-\gamma)} + m\epsilon + \gamma\Big) \\ &= 2\delta + (1-\delta)\Big(2\sqrt{\frac{k}{m}(1-\epsilon)} + 2\epsilon + \sqrt{\frac{2}{m}\ln\frac{1}{\delta}}\Big) \\ &\leq 2(\delta+\epsilon) + 2\sqrt{\frac{k}{m}} + \sqrt{\frac{2}{m}\ln\frac{1}{\delta}} \end{aligned}$$

With $k = 2^\beta$ we proved the lemma. □

Theorem 2. *Given discriminative feature f^* with $H(f^*(X)|Y) \leq \beta$, where Y takes values from $\{\pm 1\}$. Let $\mathcal{G}$ be a set of $\{\pm 1\}$-valued functions defined on $\{f^*(x) : x \in \mathcal{X}\}$. With probability at least $1-\delta$, every function g in $\mathcal{G}$ satisfies*

$$Pr(Y \neq g(X)) \leq \hat{P}_m(Y \neq g(X)) + \sqrt{\frac{2^\beta}{m}} + (\delta+\epsilon) + \sqrt{\frac{2ln(1/\delta)}{m}}$$

Proof. Apply Lemma 1 to Theorem 1. Note that we equate the δ in Lemma 1 with δ in Theorem 1.

□

Corollary 1. *Given discriminative feature f^* with $H(Y|f^*(X)) \leq \alpha$ and $H(f^*(X)|Y) \leq \beta$, where Y takes values from $\{\pm 1\}$. Given a sufficiently rich space of hypothesis space $\mathcal{G}$, the risk bound will be minimized if both α and β are minimized.*

Proof. From the risk bound in Theorem 2, the first term on the right hand side, which is the empirical risk, can be minimized (given the assumption that $\mathcal{G}$ is sufficiently rich) by minimizing $H(Y|f^*(X))$, or equivalently, by minimizing α. The second term on the right hand side can be minimized by minimizing β. □

Assumption 1 requires that there is only a finite set of "typical" feature values. This is necessary since the theorems apply to *all* possible function space. This requirement can be relaxed if we make the assumption that "typical" feature values fall within some distance ε of one of the members of $\mathcal{F}_t$, and that all functions assign the same value to each of these "typical" regions. This is illustrated in Figure 4.1.

When comparing alternative features, each satisfying $H(Y|F(X)) \leq H(Y|X) + \epsilon$, we therefore prefer the one with the smallest $H(F(X)|Y)$. Alternatively, we may aim at minimizing the

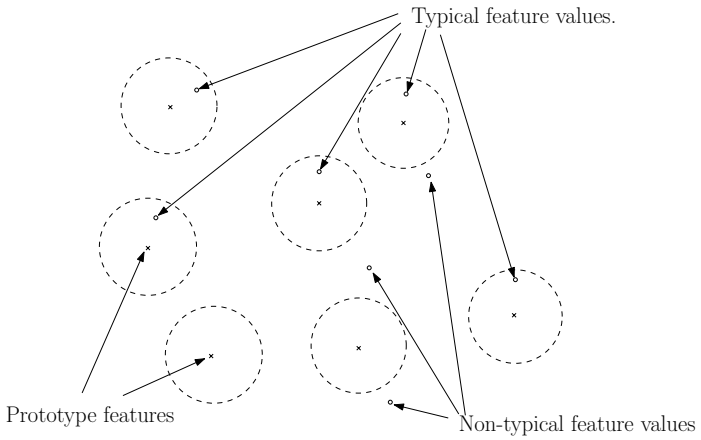


Figure 4.1: Prototype and typical feature values

following functional:

$$J(F) = H(F(X)|Y) + \frac{1}{\epsilon}H(Y|F(X))$$

Unfortunately, without additional knowledge about the underlying probability distribution, it is impossible to accurately estimate the conditional entropy empirically from the data when the training set is small. For example, when all the training examples have different X, one can simply define a feature F based on the nearest neighbor rule (itself a classifier) and empirically, with a naive estimator, achieve $\hat{J}(F) = 0$. There is no reason to believe that the feature $F(X)$ and the resulting classifier will generalize to unseen examples. In this sense, building a right feature is as hard as building the right classifier, that is, it does not work without any inductive bias.

However, it is possible that for some tasks, there exist features that are known *a priori* to have $J(F) \approx 0$. Such features capture our knowledge about patterns that are unique to a particular class of objects. What is not known, however, is a reliable way to *detect* or *extract* the feature from any unlabeled input. The problem of constructing a good feature can therefore be viewed as the problem of building a good detector for such "semantic" or "high-level" features, based on the training data. If we can ensure that the detector produces the right output *for the right reason*, i.e., it detects the intended semantic feature, then we will have high confidence that the resulting feature will generalize well.

How do we verify that a detector produces the right output for the right reason? Doing so statistically, if possible at all, will require too many training examples for many tasks. Instead, if available domain knowledge can be used to build *explanations* for the training examples, then they can be used to verify whether a feature's output is consistent with our prior knowledge. We

show how to use this idea to automatically construct features that focus on the most informative part of any input object with regard to the particular task.

4.3 Reference Features

For classification tasks, it is generally desirable that the constructed features preserve the between-class differences while eliminating as much similarity as possible. Prior domain knowledge provides a source of information for both potential differences and similarities. Our proposed algorithm exploits both differences and similarities that can be detected with feature detectors, or "sensors". We examine this from an information theoretic viewpoint.

Suppose there exists a feature transform T such that $T(X) = (X_1, X_2)$ decomposes the input X into *independent* components X_1 and X_2. Such decomposition might not be possible at the level of the raw input, but can exist at a higher abstract level. One example would be a re-parameterization of hidden variables (e.g. relative stroke parameters in handwritten characters).

Another example of such decomposition is the registration of image, where, the resulting image is independent of the transformation used to register the image. Imagine that after the digit "1" is rotated such that its main vertical stroke is at exact 90 degrees, the residual variation would be independent from the rotation itself.

The second assumption is that the first component X_1 has almost no information about the class label. In other words, it is a similarity between the classes. More precisely,

$$I(X_1; Y) \leq \epsilon$$

where I is the mutual information.

Figure 4.2 shows two very similar Chinese characters. Conceptually, the only difference between the two is an extra short-stroke in one of them. Figure 4.3 shows a decomposition where X_1 is given by the location of the two marked long strokes and X_2 is the content of the image in the region below X_1 registered against X_1.

Proposition 2. *Assuming the two assumptions above, $F(X) = T(X)_2 = X_2$ is a feature that satisfies our criteria of minimizing $H(F(X)|Y)$ while maintaining $H(Y|F(X)) \leq H(Y|X) + \epsilon$.*

Proof. We first show (the rather trivial) $H(F(X)|Y) \leq H(X|Y)$,

$$H(F(X)|Y) = H(X_2|Y) \leq H(X|Y) - H(X_1|Y) \leq H(X|Y)$$

Figure 4.2: Two very similar Chinese characters

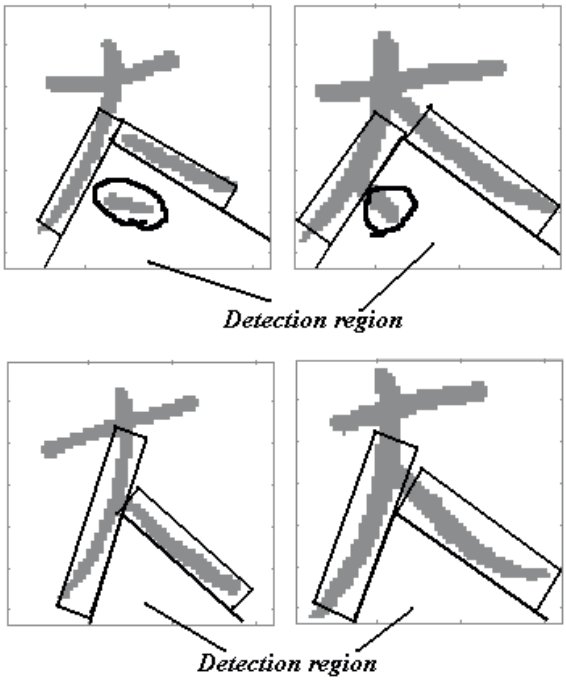


Figure 4.3: Exploiting similarity and difference with reference feature

Next, we show that $H(Y|F(X)) \leq H(Y|X) + \epsilon$,

$$
\begin{aligned}
H(Y|X) &= H(Y|X_1, X_2) \\
&= H(X_1, X_2|Y) + H(Y) - H(X_1, X_2) \\
&= H(X_1|Y) + H(X_2|X_1, Y) + H(Y) - H(X_1, X_2) \\
&= H(X_1|Y) + H(X_2|Y) + H(Y) - H(X_1, H_2) \quad \text{(independent assumption)} \\
&= H(X_1|Y) + H(Y|X_2) + H(X_2) - H(X_1, X_2)
\end{aligned}
$$

(reorder:)

$$\begin{aligned} H(Y|X_2) &= H(Y|X) - H(X_1|Y) - H(X_2) + H(X_1, X_2) \\ &\leq H(Y|X) + H(X_1, X_2) - H(X_1) - H(X_2) + \epsilon \quad \text{(assumption 2)} \\ &= H(Y|X) + \epsilon \quad \text{(independent assumption)} \end{aligned}$$

For assumption 2, we make use of the fact that $I(X_1; Y) = H(X_1) - H(X_1|Y) \leq \epsilon$ □

We name the feature detector $R(X) = T(X)_1 = X_1$ the *reference feature*, and the feature detector $F(X) = T(X)_2 = X_2$ the *target feature*. Note that this process can be repeated (and nested) as long as such decomposition is possible. The use of reference feature plays a significant role in our feature construction algorithm.

4.4 Explanation-based Feature Construction

In classical EBL, an "explanation" is a logical proof that shows how the class label of a particular labeled example can be derived from the observed inputs. But our version is weaker. We only require the explanation to identify potential low level evidence for the assigned classification label. We then use training data to calibrate and evaluate the strength of that evidence.

Our prior knowledge includes ideal stroke models of the characters of interest (roughly of the sort one obtains from a vector font) and the model of a stroke as a connected straight line of finite width.

Feature construction is performed by the following steps which we state abstractly and describe specifically in the context of the Chinese character domain.

1. **Explain each training example to obtain the association between the assigned label and the observed native features mediated by high-level domain theory concepts.** After this step, each pixel is associated with a stroke (the line that most likely made it) constrained so that the line configurations match the stroke requirements of the assigned character.

2. **Using the prior knowledge, identify 1) high-level features that are similar in both categories, and 2) high-level features that are different between the categories.** The first set form our reference strokes. These can be confidently found in new test

images since a similar stroke should be present for either instance type. The second set are information-bearing; they are character dependent.

3. **With the generated explanations, evaluate each potential similarity statistically using the training set, keeping the ones that can be detected efficiently and with high confidence.** These are strokes that are easily found in an unlabeled image and form a frame of reference to guide us to the high class-distinguishing information.

4. **Using the training examples, optimize the process of finding detection features from the reference features.** This identifies the high information image regions w.r.t. the reference strokes. The regions chosen to be larger if over the training set there is greater variance of the location of the detection strokes w.r.t. the reference strokes and tighter otherwise.

Generally, finding lines in an image is problematic. Many lines will be missed and often non-lines will be found. The process can be expensive. However, this is only done for *labeled* examples during the learning phase. Thus, we know what lines we should find and their approximate geometrical configuration. This greatly improves the line-finder's performance. But what if we can find no reference strokes? If there are no easily-found similarities between the categories, then the two classes must be very different; the classification task should be simple. Many features and algorithms should be able to differentiate them, and our knowledge-based approach is unnecessary. Next we examine this process in more detail for Chinese characters.

4.5 Classifying Handwritten Chinese Characters

Offline handwritten Chinese character recognition remains a challenging task due to the large variability in writing styles and the similarity of many characters. Approaches are either structural or feature-based [Suen et al., 2003]. The former extract strokes from a new input image and try to match the extracted strokes to the known stroke-level models. Extracting strokes is unreliable and consequently the model-matching process is problematic. Feature-based approaches utilize statistics on well-designed features and have proven to be more robust. However, the features are hand-crafted and it is not easy to exploit prior knowledge during the learning process; similar characters are difficult to differentiate using such globally-defined features.

In this work, we focus our attention to the task of distinguishing pairs of similar characters. In our approach automatically constructed features are tailored to best differentiate the characters.

Consider the pair of Chinese characters in Figure 4.4.

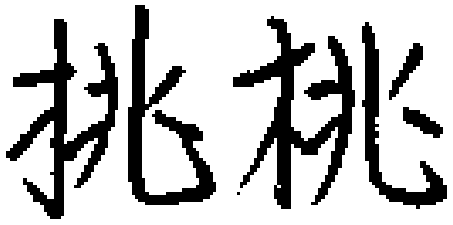


Figure 4.4: Two very similar Chinese characters

They are almost identical except the leftmost radical. Extracting a global feature that summarizes the whole character dilutes the class-relevant information concentrated on the far left of the image.

Once the informative region has been identified, there is still much variability in the *exact* location of the informative region. Figure 4.5 illustrates the variability among the first character of the pair.

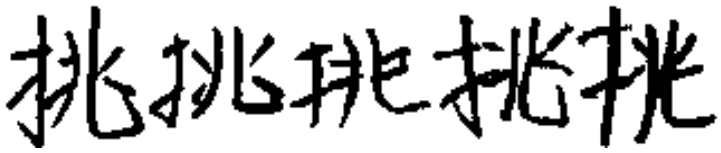


Figure 4.5: Within-class Variability

Attempting to define an absolute set of pixels (say, one 3rd of the image from the left) would result in noisy features. Too small the region we risk missing the important stroke for some of the characters, too large the region our advantage of focused feature is lost. This is where we utilize our knowledge about similarities between the two characters. The three long, roughly vertical strokes present in both characters may serve as "reference strokes." Finding them allows the target region to be more accurately identified.

4.5.1 Building Explanations

Our prior model of each character is a graph, where nodes represent a stroke and edges represent the relationship between strokes. Each stroke is itself modeled as a line segment with 5 parameters denoting its center, direction, length and thickness. Such models can either be hand-specified,

or obtained from existing character-stroke database.[1] The model need not be highly accurate as the explanation process relies primarily on its structural information. The model is used to explain each training example by finding the most likely parameters for each requisite character stroke.

In general, searching for the best set of parameters is a combinatorial problem for the general graph. This may still be acceptable since the size of the graph is small and the process is done only once for each character during training.

For efficiency, we structure these graphs into trees to employ a dynamic programming approach to produce the explanations. We use an algorithm based on [Felzenszwalb and Huttenlocher, 2000]. Our implementation uses two automatically generated trees, focusing on horizontally and vertically oriented strokes separately. Figure 4.6 shows a character model and an example explanation.

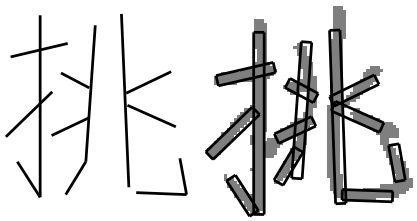


Figure 4.6: A Character Model and an Explained Example

4.5.2 Identifying Potential Similarities

Given the models for a particular pair of characters, we perform graph matching to identify strokes that are similar in terms of location, direction and length. The result of this process is the identification of a set of strokes $\mathbb{M}$ which have a match in both characters. We refer to this set as the matching set $\mathbb{M}$. These are the candidates for reference strokes. Figure 4.7 shows an example.

4.5.3 Finding Efficient Reference Stroke Detector

Any efficient feature extractor can be used in this step. Since we are concerned with lines, we use a Hough transform [Forsyth and Ponce, 2002]. In particular, we perform a Hough transform on an input image, and look for a local minimum in a specified region which reflects the variability

[1] For example, the Wen Quan Yi Project, http://wenq.org/

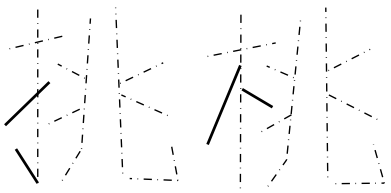


Figure 4.7: The strokes in $\mathbb{M}$ are shown as dotted lines

of the matching set stroke in the training data. We refer to this as the "Hough detector". Not every stroke in $\mathbb{M}$ can be reliably detected by the Hough detector. We use the following algorithm to select from the matching set a set of reference strokes that can be reliably detected. The explanation for each training example is used to measure how accurately the Hough detector detects a particular stroke.

1. Initialize the set of reference stroke $\mathcal{R}$ to empty
2. For each stroke S in $\mathbb{M}$
   (a) Find the range of directions and offsets for this stroke among all the training examples (from the explanations), namely, find the smallest bounding rectangle with the center $(\bar{\theta}, \bar{\rho})$, the width $\Delta\theta$, and the height $\Delta\rho$.
   (b) For each $s \in \{0.8, 1, 1.2, 1.4, 1.6\}$ and each $t \in \{0.8, 1, 1.2, 1.4, 1.6\}$
      i. Define the bounded region in Hough space as a rectangle centered at $(\bar{\theta}, \bar{\rho})$ with width $s\Delta\theta$ and height $t\Delta\theta$.
      ii. For each training example
         A. Search for the highest peak in the region
         B. Check whether the peak is within a threshold distance τ from the actual stroke orientation
      iii. Record the hit ratio $h(s,t)$ (percentage of reference strokes correctly detected using the specified parameters)
   (c) Find s^* and t^* with the highest $h(s,t)$
   (d) if $h(s^*, t^*) > h_0$ then add S to $\mathcal{R}$

The range of the detector window (s and t) in the above algorithm is chosen for simplicity. The thresholds τ (distance in Hough space) and h_0 are optimized using cross-validation.

4.5.4 Learning the final Feature

Once reference strokes are identified, the system estimates the informative region relative to the parameters of the reference strokes. We use a simple definition for our "informative region". In each character, every stroke that is not in $\mathbb{M}$ is considered a potentially informative stroke. From the explanations, we know the location of these strokes in the training examples. Using these locations, we find the smallest rectangle that includes each stroke. Whenever there are overlapping strokes, the two rectangles are combined into a single larger rectangle bounding both strokes. Figure 4.8 illustrates this.

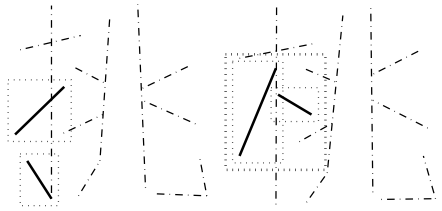


Figure 4.8: The ideal "target" rectangles

Feature points, which can be the center or endpoints of a reference stroke, receive a distance score with respect to each edge of the target window, where the score is define as $a\hat{\mu} + b\hat{\sigma}$. This combines the mean distance ($\hat{\mu}$) to the window edge and its standard deviation ($\hat{\sigma}$) given the reference strokes. The feature point with the smallest score is selected. If none of the feature points qualifies, then the edge of the image is used as the definition of the target window. Figure 4.9 illustrates this.

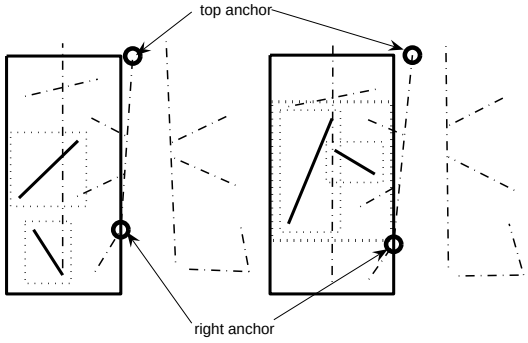


Figure 4.9: The actual "target" rectangles with respect to the feature-points. Note that there are no feature-points for the left and the bottom edge.

We assume a joint-distribution on the location of the reference strokes and the target "window" as defined by the feature-points. Each reference stroke is parameterized by the direction and the offset $R_i = (\theta_i, \rho_i)$ obtained from the Hough detector. Each target window is parameterized by 4 parameters $T = (t_1, t_2, t_3, t_4)$ which correspond to left, top, right and the bottom of the window. For example, k reference strokes and one target window form a joint-distribution on (R, T) where $R = (R_1, R_2, \ldots, R_k)$. The joint distribution is estimated from the training examples, since we know both R and T from the explanations. For unlabeled examples, we first apply the Hough detector to localize the reference strokes, $r = (r_1, \ldots, r_k)$, then find the maximum-likelihood location of the window according to the conditional probability $\Pr(T|R = r)$. Assuming that the joint-distribution is Gaussian with mean

$$\begin{pmatrix} \mu_R \\ \mu_T \end{pmatrix}$$

and covariance

$$\begin{pmatrix} \Sigma_{RR} & \Sigma_{RT} \\ \Sigma_{TR} & \Sigma_{TT} \end{pmatrix} \quad ,$$

the conditional is itself Gaussian with mean

$$\mu_{T|R} = \mu_T + \Sigma_{TR}\Sigma_{RR}^{-1}(r - \mu_R)$$

and covariance

$$\Sigma_{T|R} = \Sigma_{TT} - \Sigma_{TR}\Sigma_{RR}^{-1}\Sigma_{RT} \quad .$$

4.6 Experiments

4.6.1 ETL9B Database

We evaluate our system on pairwise classifications between difficult pairs of characters. We use the ETL9B database (a popular database of more than 3000 Chinese and Japanese characters, each with 200 examples). We first learn a literature-standard multiclass classifier using linear discriminants to identify 100 most difficult pairs (i.e. pairs of characters that result in greatest confusion). These are, as expected, pairs of very similar characters. We use the weighted direction code histogram (WDH) [Kimura, 1997] as features. These features are generally the best or among the

Pair	SVM	SVM(EBL)	Pair	SVM	SVM(EBL)
1	18.0	18.0	16	6.8	6.8
2	17.8	**17.0**	17	6.8	6.8
3	14.3	14.3	18	6.5	6.5
4	13.0	**5.8**	19	6.5	6.5
5	13.0	13.0	20	6.3	**5.3**
6	11.0	**8.3**	21	6.3	6.3
7	8.5	8.5	22	6.3	6.3
8	8.0	**7.0**	23	6.0	6.0
9	7.8	7.8	24	5.8	5.8
10	**7.5**	8.8	25	5.8	**5.0**
11	**7.5**	8.5	26	5.8	**5.3**
12	7.0	**3.0**	27	5.5	**3.5**
13	7.0	**5.8**	28	5.5	5.5
14	7.0	**2.0**	29	5.5	5.5
15	7.0	**3.8**	30	5.5	**5.3**

Table 4.1: ETL9B Error Rate (%) (5-fold cross-validation)

best for handwritten Chinese character recognition [Chen, 2005].

For each pair of characters, we learn a classifier using a linear support vector machine. We observe that the support vector machine performs significantly better than those with linear discriminants. For our system, we use the same classifier, but the input to the SVM are the WDH features extracted only from the *target* feature window found with respect to the detected reference strokes. Table 4.1 and Figure 4.10 show the results of the experiment on the 30 most challenging pairs of characters. Even though the SVM is generally robust in the presence of irrelevant features, our system managed to achieve significant improvement in many of these pairs.[2]

4.6.2 HITPU Database

In addition to the ETL9B database, we evaluated the same feature construction algorithm on the HITPU database [Shi et al., 2003], where the reported overall classification accuracy is significantly lower than that of the ETL9B. Similar to the ETL9B, the database has about 200 examples for each character. Table 4.2 and Figure 4.11 show the results for the 15 most difficult pairs in HITPU.

For most of these pairs, no reliable reference strokes can be found and therefore the entire input image is used, resulting in the same classification error rate. This is due to the larger variability and noise in the training examples (e.g. more curves), and our simple straight-line Hough

[2]We note that in several cases the examples in the database are mislabeled and our algorithm could achieve better results if these are corrected, but we decided not to modify the original database to retain its integrity.

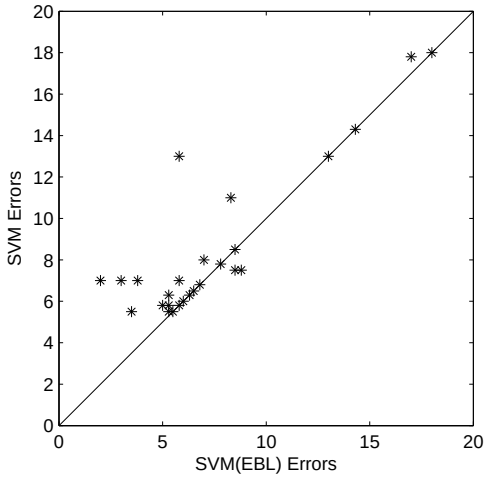


Figure 4.10: ETL9B Error Rate (%) Scatter Plot

detector is less effective at reliably detecting strokes in these inputs. However, when reliable reference strokes can be found by the algorithm, the classification performance can improve significantly.

Using a more sophisticated detector may allow more reference features to be detected reliably. Alternatively, a more complex algorithm that extracts more information from our detectors can be used. The next chapter describes one such algorithm.

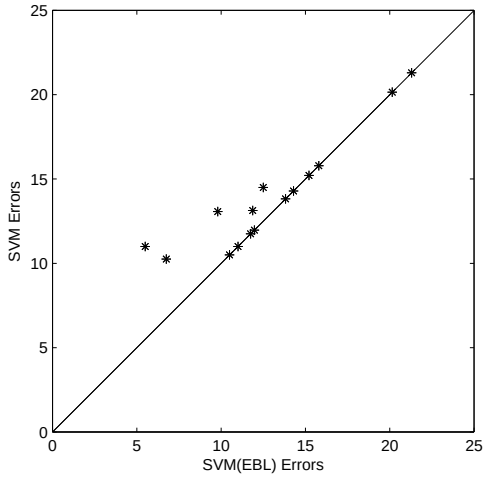


Figure 4.11: HITPU Error Rate (%) Scatter Plot

Pair	SVM	SVM(EBL)
1	21.30	21.30
2	20.15	20.15
3	15.79	15.79
4	15.21	15.21
5	14.50	**12.50**
6	14.29	14.29
7	13.82	13.82
8	13.13	**11.87**
9	13.07	**9.80**
10	11.97	11.97
11	11.75	11.75
12	11.00	11.00
13	11.00	**5.50**
14	10.50	10.50
15	10.25	**6.75**

Table 4.2: HITPU Error Rate (%) (5-fold cross-validation)

Chapter 5

Complex Discriminative Semantic Features

5.1 Introduction

Consider a binary classification task, where we wish to find a mapping from an input $x \in \mathcal{X}$ to the class label $y \in \mathcal{Y} = \{+1, -1\}$. The most general, uninformed solution space consists of all mappings $f \in \mathcal{Y}^{\mathcal{X}}$. It is well known that learning in this space is generally impossible.

From a machine learning point of view, it is generally required that we either restrict the effective solution space (i.e. hypothesis space) to a small subset of $\mathcal{Y}^{\mathcal{X}}$, or to enforce a preference among hypotheses in this space. For example, in the case where $\mathcal{X}$ consists of D-dimensional real vectors, we might restrict ourselves only to linear functions on $\mathcal{X}$. Such linear models typically are easy to handle analytically and the resulting hypothesis space has relatively low complexity compared to nonlinear models.

However, such restrictions (e.g. to linear models) are artificial and we usually have no prior belief that a good classifier can be found within the solution space, especially when dealing with real-world problems. A simple extension to this is to retain linear dependency on the *parameters* but introduce any potential nonlinearities by mapping the original input space $\mathcal{X}$ to a feature space $\varphi(\mathcal{X})$. For example, the hypothesis space $\{h|h(x) = \lambda(\mathbf{w} \cdot \varphi(x))\}$ (where $\mathbf{w}$ is the parameters to be learned, and λ is a fixed activation function for the class label) still has linear decision surfaces, although they live in the feature space $\varphi(\mathcal{X})$ instead of the original input space $\mathcal{X}$. If the feature mapping φ successfully simplifies, or "linearizes", the input space, for example, by turning each input X into a set of independent boolean variables while retaining all the information about the class label, then the hypothesis space may indeed contain the optimal solution (under the right optimality criteria). This approach however requires that the feature mapping $\varphi(\mathcal{X})$ is fixed before sending the training examples to the learning system.

It is usually difficult to come up with the "right" features in the first place. An easier option is to come up with a large set of potentially useful features and perform a feature selection proce-

dure to select the relevant ones. However, it is often statistically hard (i.e. requires large training set) to reliably select relevant features.

Instead of hand-crafting the right feature mappings for each task, we would like the learning system itself to automatically construct task-relevant features. To enable this, we need to provide additional information to the system. We would like this information to be in the form of a domain theory that captures an expert's prior knowledge about the domain, and that the same domain theory can be used by multiple, different tasks that belong to this domain. It is then up to the system to extract relevant information from the domain theory in order to construct features that are specific and relevant to individual tasks.

We propose a learning system that achieves this objective.

5.2 Explanation-Based Learning and Semantic Features

In classical EBL, one generates explanation for a given training example by deriving the class label from the observed input. Each derivation step in the explanation process is required to be based on statements given by or derivable from the domain theory. The resulting hypothesis is then required to be a generalized version of the explanation. For a particular input $x \in \mathcal{X}$ and label $y \in \mathcal{Y}$, the possible "inference paths" from x to y are restricted to those that can be derived from the domain theory. This precludes *most* of the mappings in $\mathcal{Y}^{\mathcal{X}}$ from being considered to be a hypothesis.

Given a set of training examples, the class label provides information that restricts the hypothesis space to those that are consistent with the label. Assume that the set Σ consists of all statements in the domain theory that derive the class label in one step. In other words, all hypotheses must be constructed from explanations that map the observed input to some elements of Σ. This set Σ now provides information *in addition* to the class label that further restricts the hypothesis space. The same reasoning can be recursively applied and we observe how additional information is introduced to the learning process.

From a feature construction point of view, such additional information comes in the form of *semantic features*.

For an example of semantic features, consider the pair of Chinese characters shown in Figure 5.1. To a human "expert", the main difference between the two characters is the presence of the intersection near the upper-left corner of the first character. Although this difference may be

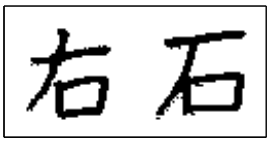


Figure 5.1: Example pair of similar characters

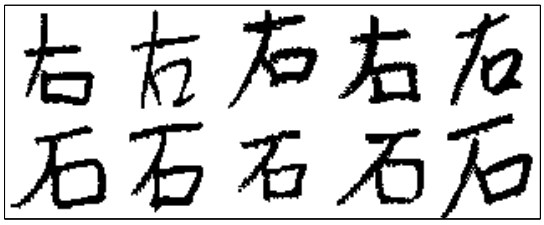


Figure 5.2: Within-class variation; each row shows 5 examples from the same character

rather obvious for a human reader, it is not easy to discover by a machine learner that parses each image as an array of pixel values, more so when there is a large within-class variation in the training examples (see Figure 5.2).

For our example pair of characters, it is rather easy to learn a classifier that correctly labels all the examples in the training set. In fact, with 200 examples per character, using raw images as input, even a linear SVM can achieve 0% training error (but with 14.25% test error). The best classifier learned using standard soft-margin SVM with Gaussian kernel, across a wide range of parameters, achieves 0% training error but only manages 11.25% test error (with 5-fold cross-validation). A human, on the other hand, can easily achieve less than 1% error rate on this pair by training on only a few examples, when the above-mentioned semantic feature has been pointed out by the teacher.

In this particular example, Σ can be the presence of the above-mentioned intersection, or simply the presence of the "protrusion" above the top horizontal stroke. This "feature" however, cannot be directly observed and cannot be discovered easily by the learner (e.g. it is not a simple linear combination of any subset of pixel values). If there is a way to insist that a hypothesis makes decision based on this feature (or an approximation to it), then we have greater belief that it will generalize well to unseen examples.

The above discussion shows how domain knowledge provides information that should help the

learning process. From a feature construction point of view, we assume that the domain theory is capable of suggesting potentially useful semantic features. However, it is still up to the learning system to construct a detector for this feature. The next section describes how we construct semantic feature detectors from available lower-level sensors.

5.3 Sensors

A feature detector, or *sensor*, is a computable mapping from $\mathcal{X}$ to a (possibly empty) set of sensor outputs:

$$S : \mathcal{X} \mapsto \mathcal{P}(\mathbb{S})$$

where $\mathcal{P}$ denotes the power set, such that each sensor output is a member of $\mathbb{S}$. Each sensor output can be a simple finite-dimensional real vector, or a structured piece of data. In all our examples, the sensor outputs are either pixel values (e.g. image patches) or geometrical objects such as line segments, edge segments etc. Although we do not formally impose any requirements in terms of computational cost, we assume that a sensor can produce its output in polynomial time (e.g. polynomial in terms of the size of input images).

We differentiate between the term "feature" and "sensor". A *feature* is a property of the input object that is abstract and its value can either be computed from the input itself or simply be given by an expert. A sensor, on the other hand, is strictly a computable function on the input. It is possible to define a feature based on the output of a sensor (its values are then simply the sensor outputs). We refer to such features as "low-level" features, in contrast with "semantic" or "high-level" features, whose values are typically not directly computable, but can sometimes be approximated by sensors.

The object category, or the class label itself, can therefore be viewed as a feature (as assigned by an expert). We can think of this feature as the ultimate, highest-level feature for our task. The solution $f \in \mathcal{Y}^{\mathcal{X}}$ that we seek is therefore simply a feature detector for this feature. From this point of view, learning a classifier is equivalent to constructing a semantic feature detector.

At the other extreme, the raw input (e.g. pixel values) can be regarded as the lowest level features. Working directly at this level is often undesirable since the class label is usually a complex, nonlinear function of the raw input. Intermediate-level features, which consist of simple, but may be nonlinear functions of the input can be much more informative and easier to work with. Our approach focuses on the use of such features in the forms of sensors and their corresponding "in-

terpretations" to construct more complex semantic feature detectors.

5.3.1 Interpreting Sensor Outputs

In order to support reasoning with features and sensors, we need a mechanism to relate (low-level) sensor outputs with higher-level features. We allow such relationships to be expressed both qualitatively and quantitatively in the domain theory.

The qualitative part of the domain theory defines *abstract objects* that can be associated with sensor outputs. An abstract object is a named conceptual entity with properties. For example, a stroke in a handwritten character is an abstract object. The properties of a stroke may include its length, curvature etc. An abstract object can also be part of another abstract object. For example, a stroke object intersecting with another stroke can be decomposed into two stroke segments, where each stroke segment can be a separate abstract object. Similarly, a "handwritten character" object consists of several stroke objects. We do not formally restrict the ways in which abstract objects are represented in the domain theory, as long as each has a unique name. For example, each stroke in a character has a unique name, such as C17 or S21.

Each abstract object can have an associated *appearance model*. An appearance model defines a parameterized space of configurations. Each instance of the abstract object can have a different configuration. For example, a stroke C17 can be modeled as a line segment with a fix width, with parameters specifying the location of the two endpoints and the width. Different instances of C17 have different configurations, due to variations in handwriting.

Each sensor output can be associated with one or more abstract objects. The associated object is considered a potential interpretation of the sensor output. In other words, the sensor output can be used to infer the configuration of the associated object. For example, suppose there exists a sensor S that can extract straight line segments from any image. Suppose a given image x is a handwritten example of a character that contains a stroke C17. Then a particular sensor output (i.e. a line segment) $s_0 \in S(X)$ can be associated with C17 (or "interpreted" as C17). It is up to the domain expert to determine whether s_0 is correctly interpreted, or in other words, whether it is consistent with this particular interpretation.

The quantitative part of the domain theory provides the means to evaluate the consistency between a sensor output and an abstract object. Associated with each interpretation is a set of *consistency metrics*. Each consistency metric is a function that maps a sensor output to a non-negative real value. This value is a quantitative assessment of the sensor output in terms of the

amount of "deviation" from the ideal configuration of the abstract object. Each consistency metric addresses a particular aspect of the object. For example, suppose stroke C17 is supposed to be completely straight and vertical in its ideal configuration. A possible consistency metric can measure the angular difference between a given line segment and the ideal vertical direction. Another metric can measure the curvature of the line segment.

Similarly, consistency metrics can also be associated with a pair (or more generally a set) of interpretations. For example, two sensor outputs s_0 and s_1 can be interpreted as objects C17 and C3 respectively. Suppose C17 is known to intersect with C3, then a consistency metric that measures how much s_0 and s_1 "intersect" can be defined.

We emphasize that although the consistency metrics provide quantitative assessment to interpretations, their numerical values are not required to be accurate or absolute. It is often the case that our domain knowledge can only make confident assessments qualitatively, but not quantitatively. For example, how do we set the boundary for "being vertical"? These are uncertainties that not even an expert can easily handle and it is usually better if they are resolved using the actual training data. We only require that the consistency metrics allow comparisons between competing interpretations in a relative sense, and as long as extreme cases (e.g. very unlikely interpretation) can be confidently ruled out, the metrics will remain informative even if they are noisy.

5.3.2 Consistency Metrics

We now provide the precise definition for the form of consistency metrics used in this work. The consistency metrics serve two purposes. First, they provide a consistent way to linearize any properties of abstract object parts and allow simple combination with other properties. Secondly, they allow uncertainties to be resolved statistically.

For the first purpose, we define a transform that normalizes the range of each metric to between 0 and 1. For the second purpose, we associate a tunable parameter with each metric. Recall that each abstract object, when used as an interpretation for a sensor output, is associated with a set of consistency metrics. Denoting the abstract object as I and the sensor output as s, let $M^I = \{m_j^I | j = 1, \ldots, k\}$ be the set of consistency metrics for I. Each m_j^I is defined as a function in the following form:

$$m_j^I(s) = \sigma\big(\theta_j^I - \frac{1}{T} d_j^I(s)\big)$$

where σ is the standard logistic sigmoid function and θ_j^I is the tunable parameter. T is a fixed constant that sets the "steepness" of the sigmoid function and d_j^I is the fixed non-negative "devi-

ation" function associated with the metric m_j^I. Qualitatively, we want m_j^I to be large when d_j^I is small (i.e. small deviation from ideal implies good consistency). Although T can be made into a tunable parameter, but fixing it will disallow the sign change of the gradient of the sigmoid function, preserving the intended relationship between m_j^I and d_j^I. The deviation functions d_j^I are also assumed to be bounded and normalized to a range between 0 and 1. This enables a more uniformly controlled range for the tunable parameters θ_j^I. Figure 5.3 shows the form of the function with varying values of θ_j^I and a fix T.

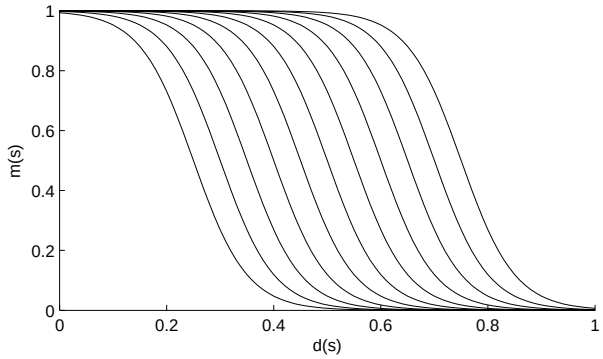


Figure 5.3: Consistency metric functions for various values of θ

5.3.3 Induced Sensors

We have previously defined a sensor to be a mapping from the input to a (possibly empty) set of sensor outputs. For example, an edge detector can be used as a sensor that extracts all significant edges from an input image (with a fixed threshold for significance).

In this work, we use sensors mainly as detectors for abstract object parts. Each sensor, when applied to the input, has an intended "target object" I_t. Since the sensor may actually detects multiple outputs, it is computationally costly (and not strictly necessary) to keep every sensor output. Instead, we simply select the sensor output that is the most likely to be the intended object. As we will see later, this choice needs not be correct and a simple way to make a choice is to use the deviation functions $d_j^{I_t}$ in the consistency metrics of the target object. For each sensor output s, the "total deviation" $d_{\text{total}} = \sum_{j=1}^{k} d_j^{I_t}(s)$ is computed and the one with the smallest d_{total} is selected. This d_{total} can be refined, for example, by weighting each $d_j^{I_t}(s)$ differently and adaptively in order to improve the accuracy of the sensor with respect to the target object. We do not pursue this direction, since it does not provide true improvement to the solution as there

are uncertainties in sensor interpretations that cannot be resolved this way. Instead, we avoid this extra complication and employ a different strategy to handle the inherent imperfection in our sensors.

These sensors are said to be induced by the intended target object, and they are biased towards detecting sensor outputs that are more likely to be consistent with the target object. We note that, using this strategy, significantly different sensors can be derived from the same basic sensor by associating different target objects with the basic sensor.

From this point on, we assume that every sensor is an induced sensor, and always produces exactly one sensor output. In the case where an empty set is detected, we assign a special dummy value as its output. When a sensor S is applied on an input x, we may now denote its (single) output as simply $S(x)$, with $S(x) \in \mathbb{S}$.

5.4 Features

Consider a sensor S and an interpretation I of its output (as an abstract object part). This *(S,I)* pair can itself be considered a feature. Associated with the interpretation I is a set of consistency metrics m_j^I, $j = 1, \ldots, k$. As defined in the previous section, each metric m_j^I is a non-negative, real-valued function on the sensor outputs:

$$m_j^I : \mathbb{S} \mapsto \mathbb{R}$$

We can think of this set of functions as a vector-valued function that generates a k-dimensional feature vector for each sensor output:

$$M^I : \mathbb{S} \mapsto \mathbb{R}^k$$

This feature vector can then be used to evaluate how consistent a particular sensor output is with interpretation I. Note that k needs not be the same for every interpretation.

Similarly, a pair of sensors S_1 and S_2, together with a corresponding pair of interpretations I_1 and I_2 are associated with a set of consistency metrics $m_j^{I_1,I_2}$, $j = 1, \ldots, k$. We then have M^{I_1,I_2} as a function that generates k-dimensional feature vectors from sensor output pairs. This feature vector evaluates how consistent a given pair of sensor instances is with the *relationship* between the two abstract object parts.

Given a *(sensor,interpretation)* pair, we can construct a decision function that outputs the be-

lief (or likelihood) that a given sensor output is indeed correctly interpreted (as the object part). Since we rely on the individual consistency metric to handle any necessary non-linearities, we simply use a sigmoid function over a linear combination of components of the feature vector for this belief function:

$$B^I(s) = \sigma\big(w_0^I + w_1^I m_1^I(s) + \cdots + w_k^I m_k^I(s)\big) = \sigma\big(\mathbf{w}^I \cdot M^I(s)\big)$$

where $\mathbf{w}^I = \langle w_0^I, \ldots, w_k^I \rangle$ are tunable parameters, and σ is the function:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \qquad .$$

Note that B^I also depends on the parameters for individual consistency metrics $\theta^I = \langle \theta_1^I, \ldots, \theta_k^I \rangle$.

5.4.1 Combining Sensors

Armed with the belief function B^I, one may come up with a general probabilistic model that can be used as a classifier. We first describe one such model then discuss the problems associated with it and how it motivates our final solution.

Suppose the class membership of a given input object largely depends on the presence of certain sets of abstract object parts. Suppose also that we have at our disposal a set of sensors, where each sensor is capable of detecting some of the object parts. However, given the set of sensor outputs obtained from an input it is unclear what the correct "joint-interpretation" for the entire set of sensor outputs is (i.e. what part does each sensor actually detect for this particular input?).

We address this uncertainty by modeling a joint distribution over the possible *interpretations* for each sensor output. Let X and Y be the random variables for the input and the class label respectively. Let S_i, $i = 1, \ldots, m$ be the sensors. Let $\mathbf{H} = (H_i)$, $i = 1, \ldots, m$ be the random variables for the interpretation of each corresponding sensor. Each H_i takes values from the (finite) set of abstract object parts $\mathcal{I} = \{I_1, \ldots, I_n\}$. Note that not all object parts can be detected by all sensors, so each H_i typically takes its values from only a subset of $\mathcal{I}$.

We model the *conditional distribution* over $(Y, \mathbf{H})$ given X with a *conditional random field* as follows:

$$P(Y, \mathbf{H}|X; \Theta) = \frac{1}{Z} \prod_C \psi_C(Y, \mathbf{H}, X; \Theta)$$

where Θ is the model parameter, Z is the normalization factor, and C ranges over single nodes (i)

or pairs of nodes (i, j). Each ψ_C is a potential function, to be defined below.

Each node i in the graph for the random field corresponds to the interpretation H_i for sensor S_i. For pairs of interpretations where consistency metrics are defined, the corresponding nodes are connected. We refer to the pairwise interpretations for nodes i and j as $H_{(i,j)} = (H_i, H_j)$ and the sensor outputs $S_{(i,j)} = (S_i, S_j)$. Assuming that we only have consistency metrics (and therefore the belief function) for either single-nodes or pairs of nodes, then C can either be a single node (i) or a pair of nodes (i, j). Each ψ_C is a potential function for the corresponding interpretation H_C (single or pairwise), and is defined based on the belief functions B^{H_C} as follows:

$$\psi_C(Y, \mathbf{H}, X; \Theta) = \exp\{u^{H_C} B^{H_C}(S_C(X)) + v^{Y,H_C}\} \tag{5.1}$$

The function B^{H_C} is exactly as defined in the previous section and it depends on parameters $\mathbf{w}^{H_C}$ and θ^{H_C} (as defined in the previous sections). The scalar weights u^{H_C} and v^{Y,H_C}, together with $\mathbf{w}^{H_C}$ and θ^{H_C} for all Y and H_C form the entire set of model parameters Θ.

The belief functions evaluate the consistency between interpretation H_C and the sensor outputs S_C. The "bias" v^{Y,H_C} evaluates the consistency between the abstract objects associated with H_C and the class Y.

Given an input x, classification can be performed by selecting y with the highest marginal probability $P(y|x; \Theta)$, given by:

$$P(y|x; \Theta) = \sum_{\mathbf{H}} P(y, \mathbf{H}|x; \Theta)$$

It is possible to learn the model parameters Θ from a training set of examples, by defining an appropriate objective function. For example, given N training examples (x_i, y_i), $i = 1, \ldots, N$, we can find the maximizer for the regularized log likelihood of the data:

$$L(\Theta) = \sum_{i=1}^{N} \log P(y_i|x_i; \Theta) - \gamma \|\Theta\|^2$$

where γ is a scalar weight for the (zero-mean Gaussian) regularizer. This is most easily done when the correct interpretation for each sensor $\mathbf{H}$ is available for the training set, since there is no need to marginalize over all possible interpretations $\mathbf{H}$ when calculating the log likelihood. Otherwise, the random field becomes a *hidden conditional random field* (see [Quattoni et al., 2007]) and learning can be more costly and noisy. In either case, inference (e.g. performing the classification) can

be computationally intractable since the marginalization over $\mathbf{H}$ is exponential in terms of the largest clique size of the graph. Restricting the graph to a tree, on the other hand, can seriously limit the information that we can incorporate into the model.

5.4.2 A Markov Logic Interpretation

Our way of using the belief functions in the conditional random field can be viewed as a generalized Markov logic network (see [Richardson and Domingos, 2006]). In first-order Markov logic networks, each node represents a ground predicate, and each clique represents a formula. Each formula is associated with a potential function similar to the one defined in Eq. 5.1, with the belief function B^{H_C} replaced by the activation function of the corresponding formula. The activation function has a value 1 if the formula is satisfied, and 0 otherwise. Our belief function, on the other hand, takes real values between 0 and 1. It plays the role of the logical formula, where its "activation" is determined by the level of consistency between the sensor outputs and the assigned interpretations.

As with Markov logic networks, we can therefore view our model as a way of combining both logical and statistical inference. A logical "inference path" in our case corresponds to a set of interpretations that jointly achieves high consistency with the sensor outputs. This special set of interpretations is in fact, simply the semantic feature that we try to detect with our sensors. This motivates the design of our final solution.

5.4.3 Semantic Features

We assume that each semantic feature requires a set of object parts under certain configurations as *evidence* for its presence. When a sensor instance is paired with (i.e. interpreted as) each of the required object parts, this group of sensors becomes a potential detector for this semantic feature.

Recall that each (S_C, H_C) pair is itself a (lower level) feature, whose feature vector is generated by the associated set of consistency metrics. The overall consistency of this interpretation is given by the belief function $B^{H_C}(S_C)$. We use both single ($C = (i)$) and pairwise ($C = (i, j)$) interpretations.

Consider a subset of d connected nodes $\mathbb{I} = \{i_1, \ldots, i_d\}$ in the graph of the conditional random field, with a *particular* interpretation assigned to each $i \in \mathbb{I}$, such that $H_i = h_i$ for each $i \in \mathbb{I}$. Recall that $h_i \in \mathcal{I}$ for all $i \in \mathbb{I}$. Let $I_{\mathcal{F}}$ denote the set of all node indices (single and pairwise) among nodes in $\mathbb{I}$ where there exists a corresponding belief function. The set of all interpreta-

tions $\{h_C | C \in I_{\mathcal{F}}\}$ constitute the necessary abstract objects in the semantic feature (or"inference path") $\mathcal{F}$. Instead of the log-linear model in the conditional random field, we use a sigmoid function to combine the set of belief functions for $I_{\mathcal{F}}$ and define the likelihood function for $\mathcal{F}$ as:

$$L^{\mathcal{F}}(x) = \sigma\left(v^{\mathcal{F}} + \sum_{C \in I_{\mathcal{F}}} u^{h_C} B^{h_C}\left(S_C(x)\right)\right)$$

where, again, $v^{\mathcal{F}}$ and u^{h_C} are the parameters.

Let F be a Bernoulli random variable, with value 1 if feature $\mathcal{F}$ is present and 0 otherwise. We can view $L^{\mathcal{F}}$ as a conditional likelihood function for F, i.e., $P(F = 1|X) = L^{\mathcal{F}}(X)$ and $P(F = 0|X) = 1 - L^{\mathcal{F}}(X)$.

5.4.4 Induced Sensors for Semantic Features

Recall the notion of induced sensors (see section 5.3.3), where each sensor is constructed with respect to an intended target object. We extend this notion to a sequence of sensors that is constructed with respect to an *ordered set* of intended target objects.

The first sensor in this sensor sequence selects its sensor output based on the consistency metrics for its intended interpretation. The second sensor, however, uses both the consistency metrics for its intended interpretation *and* the consistency metrics for the pairwise interpretation between the first two sensors. In this manner, each sensor in the sequence makes use of consistency metrics for all pairwise interpretations between itself and each of its "parent" sensors, in addition to the interpretation for its own intended target object.

Alternatively, we can view each sensor in the sequence as a *conditional* sensor, where its output is influenced by a particular interpretations of the outputs from previous sensors. Such a sequence of sensors can be much more effective in detecting a set of related abstract objects compared to a set of sensors that act independently.

5.5 Sensor Tree

Consider a classification task. Suppose, according to the domain theory, that a semantic feature $\mathcal{F}$ is unique to the members of one class. We refer to this class as the *target class*.

In an ideal case, feature $\mathcal{F}$ can be reliably detected by a sequence of sensors. In other words, there exists a very discriminative likelihood function $L^{\mathcal{F}}$ for $\mathcal{F}$. To classify an input x, this se-

quence of sensors is applied to x and if $L^{\mathcal{F}}(x) > 0.5$ then x belongs to the target class.

Finding such a set of sensors and learning the corresponding parameters can be difficult for many reasons. First, there exists inherent noise in a sensor's output such that it cannot always detect the intended target object. Secondly, there might be inherent noise in the input in the sense that two different abstract objects with similar appearance can be mistaken for one another. Lastly, the same semantic feature may appear differently in different class instances and a single sensor sequence cannot handle all such variations.

As an illustration, consider the two examples in Figure 5.4, both from the same character. The black line segments show sensor outputs that are similar, but in fact representing different strokes. For this character, there may not exist a single sensor that can consistently detect either the top or the bottom horizontal stroke.

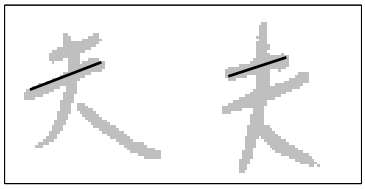


Figure 5.4: Similar sensor outputs from different objects

Instead, consider the following strategy of applying sensors. Assume that I_1 and I_2 refer to the top and bottom horizontal strokes of the character in Figure 5.4 respectively. First, a sensor S_1 is applied, looking for I_1. Then, a second sensor S_2 is applied, looking for I_2, *conditioning* on the interpretation of S_1's output as I_1. Next, a third sensor S_3 is applied, looking for I_1, conditioning on the interpretation of S_1's output as I_2. This setup of sensors can be represented by a tree as shown in Figure 5.5.

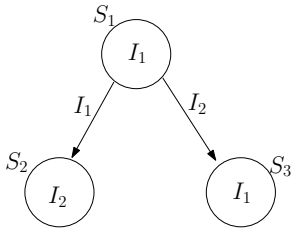


Figure 5.5: A simple sensor tree

In any case, at most one of the sensor sequences (either (S_1, S_2) or (S_1, S_3)) can have correct

joint-interpretations. We note that although both S_1 and S_3 look for I_1, S_3 is a different sensor since, given S_1's interpretation, it will have higher preference for line segments located above S_1's output (and could fail if S_1 had correctly detected I_1). We believe that, as long as the domain theory is adequate, the two sensor paths will have likelihood functions that allow easy identification of the correct interpretations.

The above example shows that, instead of relying on a single well-tuned sensor sequence, we can combine sensors into a tree structure such that uncertainties in sensor interpretations can be resolved by applying additional sensors.

5.5.1 Neural Network for the Sensor Tree

Assume that a sensor tree is created for detecting the semantic feature $\mathcal{F}$. By definition, the presence of $\mathcal{F}$ depends on a particular set of abstract objects being detected. The structure of a sensor tree is a directed acyclic graph with a specific node designated as the root. Each path of the sensor tree (from the root node to a leave node) represents a sequence of sensors with a corresponding sequence of intended interpretations that is considered sufficient for detecting $\mathcal{F}$. We will deal with the issue of what is considered "sufficient" in later sections.

Let $\mathbb{I} = \{1, \ldots, m\}$ be the set of node indices of the sensor tree with m nodes. The corresponding sensor for node i is denoted S_i. Each (directed) edge in the tree from node i to j is associated with an interpretation for sensor S_i. We use $p = 1, 2, \ldots$ as the index for each *path* in the tree (no particular order). Given the path index p and a node i, there is at most one outgoing edge from i that belongs to path p. The associated interpretation for S_i in path p (if it exists) is designated h_i^p.

Associated with each path is a likelihood function that evaluates how much we believe that the sensors in the path have the correct interpretations. This likelihood in turn depends on the belief functions associated with each interpretation in the path. We define I^p to be the set of all node indices (both single and pairwise) in path p. For example, if path p has the node sequence (i_1, i_2, i_3), then $I^p = \{i_1, i_2, i_3, (i_1, i_2), (i_1, i_3), (i_2, i_3)\}$. We also use the shorthand $h^p_{(i_1,i_2)}$ for $(h^p_{i_1}, h^p_{i_2})$ and $S_{(i_1,i_2)}$ for (S_{i_1}, S_{i_2}). The likelihood function for path p is defined as:

$$L^p(x) = \sigma\left(v^p + \sum_{C \in I^p} u^p_C B^{h^p_C}\big(S_C(x)\big)\right) \tag{5.2}$$

Given a sensor tree, we naturally expect that some sensors share the same interpretation. Fur-

thermore, any two paths can have some common sensors, possibly with common interpretations. This information sharing can be captured by representing the likelihood functions as a neural network with multiple outputs.

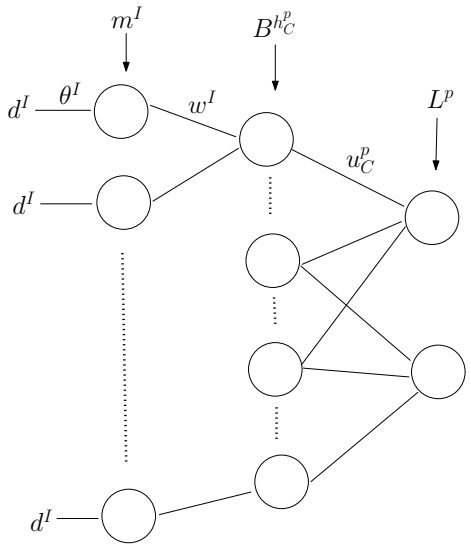


Figure 5.6: Neural network architecture for sensor trees

Figure 5.6 shows the neural network architecture for a sensor tree and identifies the parameters and activation functions for each layer. Both the consistency metrics and the belief functions are activation functions for hidden nodes in the neural net. All activations use the logistic sigmoid functions.

We enforce weight sharings at the level of belief functions (the w^I's and θ^I's) by forcing all sensors with the same interpretation to use the same weights for the shared belief functions. For example, for the sensor tree in Fig. 5.5, the belief functions for interpretation (S_1, I_1) and (S_3, I_1) use the same weights (for the consistency metrics associated with I_1). The use of weight sharings in this manner not only reduces the capacity of the neural network, but does it in a way that is consistent with the domain theory.

Given that the sensor tree is designed to detect a specific semantic feature $\mathcal{F}$, the likelihood functions L^p are used jointly to indicate not only the presence of $\mathcal{F}$ but also the sensors that are responsible for the detection. This allows us to evaluate not only how well the sensor tree works, but whether it works for the right reason.

5.5.2 Computational Complexity

With the conditional random field, evaluating the likelihood function requires the marginalization over possible interpretations of all sensor outputs, which is intractable in general (even if we restrict ourselves to only single and pairwise interpretations).

With the neural network, the computational complexity is dominated by the size of the sensor tree. For any paths, the number of belief functions (potential functions) to evaluate is quadratic in terms of the path length since we use only single and pairwise interpretations. So, it is straightforward to show that the computational complexity is only polynomial in terms of the tree size.

To gain insight into the difference between the two approaches. Consider looking for the two horizontal strokes in Figure 5.4. With a conditional random field, one applies the "horizontal stroke detector" to obtain the set of sensor outputs (assume that they are all line segments that are more or less horizontal). Associated with each sensor output is a node in the random field. Each sensor output can have several interpretations (e.g. whether it is the top or the bottom horizontal stroke). The inference process has to essentially evaluate each "joint-interpretation" over the sensor outputs. It is easy to see that for features that consist of many parts, the computational cost grows exponentially in the number of parts.

With the sensor tree (e.g. Figure 5.5) each sensor is associated with the target object and the major computational cost comes from the fact that sensors deeper in the tree need to be evaluated against each "parent" sensor. In the most unfortunate case, *every* detection order needs a separate path in the tree and the resulting tree will be large. However, due to sharing of information, we typically expect the number of paths to be much smaller (less than exponential), and this depends on the domain theory.

5.6 Learning Algorithm

We propose an iterative algorithm to construct and refine sensor trees. In each iteration, a new sensor tree is created and its parameters optimized. Its performance on the training set is then evaluated and the errors analyzed. The error evaluation updates a "confusion database", which is then used in the next iteration to create a new sensor tree. The followings are the major steps in the algorithm:

1. Find a target semantic feature $\mathcal{F}$.

2. Construct a sensor tree for $\mathcal{F}$ using the current confusion database.

3. Apply the sensor tree to the training examples (to obtain sensor instances) and obtain the target labels.

4. Optimize sensor tree parameters.

5. Evaluate the sensor tree performance on the training examples.

6. Analyze errors and update the confusion database

7. If stopping condition is not met, go to step 2.

Multiple sensor trees can be created, each targeting a different (or even the same) semantic feature. The final classifier is simply an ensemble of sensor trees.

5.6.1 Target Semantic Feature

Each sensor tree is constructed and optimized so that it is a reliable detector for a designated target semantic feature. For a classification task, this target feature is a proposed "difference" between object classes or a unique pattern in an object class. We rely on the domain theory to come up with such potential features.

In the previous chapter, we proposed a method of finding such features by comparing prototype characters. The same reasoning method applies to here.

For binary classification tasks, a target feature $\mathcal{F}$ specifies two matching sets $\mathcal{R}_1^{\mathcal{F}}$ and $\mathcal{R}_2^{\mathcal{F}}$ of abstract objects in each of the object class. These constitute the reference features. In addition, a target object $\mathcal{T}^{\mathcal{F}}$ unique to one class is specified. The reference features can be seen as setting up a "frame of reference" since they are expected to exist in either class. Given the reference features, the target object $\mathcal{T}^{\mathcal{F}}$ is expected have a distinctive appearance such that an appropriate sensor can be used to detect its presence (or absence) reliably. Figure 5.7 shows an example.

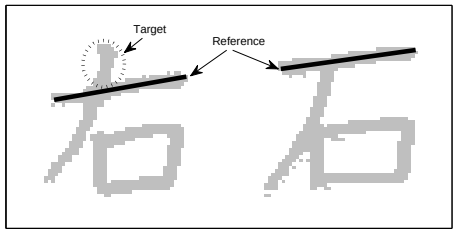


Figure 5.7: Example target and reference features

5.6.2 Constructing Sensor Trees

We assume that every sensor is an induced sensor, and is therefore associated with an intended abstract object. Additionally, a set of conditional objects can be specified — these correspond to any previous sensor outputs. We also assume that a procedure GET_SENSOR is available such that given a target object, together with any conditional objects, one or more sensors can be obtained. We also allow GET_SENSOR to fail in case there is no available sensor for the target object.

The basic idea in constructing a sensor tree is to identify possible interpretations for a given sensor, conditioning on the interpretations of all the previous sensors in the same sequence (i.e. path). In the ideal case, every sensor, when applied, will detect precisely the intended target object and nothing else. In this case, the only interpretation is the target object. In general, however, we expect that a set of objects can be detected.

The set of possible interpretations for a given sensor can be constructed based on the prior information in the domain theory as well as actual observations on the training examples.

- Obtaining possible interpretations from domain theory

  In the most uninformative case, we can simply assume that the only possible interpretation for a sensor is the target object. In this case, we rely solely on empirical observations to identify possible interpretations.

  Alternatively, given a sensor S, we can use the consistency metrics of its intended interpretation I to find other possible interpretations. Recall that associated with interpretation I is a set of consistency metrics $M^I = \{m_j^I | j = 1, \ldots, k\}$ with parameters θ_j^I. We define a default value for each θ_j^I.

  For a candidate interpretation, we generate a sensor output for that object (e.g. by using a prototype class example) and compute each m_j^I. The candidate interpretation becomes a potential interpretation if $m_j^I > 0.5$ for all j.

- Obtaining possible interpretations empirically

  This can be done after the sensor tree has been applied to the training examples. The explanation for each example provides the correct (or most likely) interpretation for each sensor output. A *confusion database* can be used to keep track of all interpretations that have been actually observed in the training examples. Each entry in the confusion database specifies

the sensor's intended object, all its conditional objects, and gives the list of objects that can be detected by this sensor.

We assume that a procedure GET_INTERPS(S) returns all possible interpretations for sensor S using one or both of the above methods. Note that although all intended interpretations are objects from the *target class*, we keep track of interpretations from both classes of objects.

We assume that the set $\mathcal{I}_1$ contains all abstract objects in the target class that are of interest for this feature (and similarly $\mathcal{I}_2$ for the non-target class). The sets $\mathcal{I}_1$ and $\mathcal{I}_2$ may initially contain simply the objects in $\mathcal{F}$ but will grow after observing new objects being detected by the sensors in the training examples.

The following algorithm constructs a new node in a sensor tree for feature $\mathcal{F}$. It can be recursively applied until the tree is complete. The tree is complete when every path includes all the reference objects in $\mathcal{R}_1^{\mathcal{F}}$ as well as the target $\mathcal{T}^{\mathcal{F}}$, and that the final leave node does not have any interpretation from the non-target class.

1. Create a new tree node i. Let I_p be the set of all interpretations for sensors in the path leading to i.
2. For each $I \in \mathcal{I}_1$
   (a) Skip I if $I \in I_p$
   (b) $S^I \leftarrow$ GET_SENSOR(I)
   (c) $J^I \leftarrow$ GET_INTERPS(S^I)
   (d) Calculate the total number of possible interpretations, Score(I) $\leftarrow |J^I|$
3. MinScore $\leftarrow \min_I$ Score(I)
4. $I_{min} \leftarrow \{I \,|\, \text{Score}(I) \leq \text{MinScore} + \tau\}$
5. If $\mathcal{R}_1^{\mathcal{F}} \subset I_p$ and $\mathcal{T}^{\mathcal{F}} \in I_{min}$ then $T_i \leftarrow \mathcal{T}^{\mathcal{F}}$
   (a) Otherwise if there exists $I \in \mathcal{R}_1^{\mathcal{F}} \bigcap I_{min}$ such that $I \notin I_p$ then $T_i \leftarrow I$
   (b) Otherwise choose any $I \in I_{min}$ and set $T_i \leftarrow I$
6. $S_i \leftarrow S^{T_i}$
7. Create a new edge for each $I \in J^{T_i} \bigcap \mathcal{I}_1$

5.6.3 Parameter Optimization

After the sensor tree is constructed, it is applied to the training examples. We assume that the explanation (i.e. the correct interpretation) for each sensor output can be obtained (see the previous chapter). Using this information, it is straightforward to determine for each training example in the target class, the path that has the correct interpretations for all its sensor outputs (or none).

For each training example, we assign a target label t^p for every path p in the sensor tree. For examples in the target class, t^p for is 1 for the correct path and 0 for all other paths. For examples in the non-target class, all t^p is set to 0.

The likelihood function for path p, L^p is as defined in section 5.5.1. The corresponding neural network for the sensor tree has parameters $\Theta = \langle \Theta^p \rangle$ for all path p where each Θ^p is given by $\Theta^p = \langle v^p, u^p_C, \mathbf{w}^{h^p_C}, \theta^{h^p_C} \rangle$ for all C in path I_p.

Given N training examples $(x_i, y_i), i = 1, \ldots, N$, together with the labels $t^p_i, i = 1, \ldots, N$ for each path p, we find the maximizer for the following regularized log-likelihood of the data:

$$L(\Theta) = \sum_{i=1}^{N} \sum_{p} \log P(t^p_i | x_i; \Theta) - \gamma \|\Theta\|^2$$

Note that we assume each path label to be independent from the others. This is certainly not true but a reasonable approximation and allows consistent handling of training examples from both the target and non-target class. Using the likelihood function L^p (as defined in equation 5.2), we can rewrite $\log P(t^p_i | x_i)$ as:

$$\log P(t^p_i | x_i) = t^p_i \log L^p(x_i) + (1 - t^p_i) \log(1 - L^p(x_i))$$

and therefore the regularized log likelihood as:

$$L(\Theta) = \Big(\sum_{i=1}^{N} \sum_{p} t^p_i \log L^p(x_i) + (1 - t^p_i) \log(1 - L^p(x_i)) \Big) - \gamma \|\Theta\|^2$$

$L(\Theta)$ is a smooth function, and its gradient can be efficiently evaluated using back-propagation. However, this objective function is in general non-convex so we do not expect to find the global maximum. Any optimization algorithm can be used to find a (locally) optimal Θ. In our experiments we use the BFGS Quasi-Newton method which only needs the gradient information.

5.6.4 Evaluating the Training Examples

For any input x, it is straightforward to compute the likelihood function $L^p(x)$ for each path p of the sensor tree. Recall that each path represents a particular interpretations for the sensor sequence in that path and that, intuitively, at most one path should be the "correct" path. A simple strategy to decide the final label for x is to assume that the feature $\mathcal{F}$ is present in x if there exists a path p with $L^p(x) > \varphi$ (e.g. $\varphi = 0.5$), and that $\mathcal{F}$ is not present otherwise. The value of φ can be optimized based on the training example, and can even be different for each path. This strategy assumes a positive detection for $\mathcal{F}$ even when there is more than one p with $L^p(X) > \varphi$.

Alternatively, one could adhere to the independence assumption and use an entirely probabilistic evaluation. Let the random variable F_p be a boolean variable with value 1 if feature $\mathcal{F}$ is detected in path p and 0 otherwise. Each F_p has independent (conditional) Bernoulli distribution $P(F_p = 1|X) = L^p(X)$. Then the probability that $\mathcal{F}$ is present in x is simply:

$$P(\mathcal{F} \text{ present}) = 1 - P(\mathcal{F} \text{ not present}) = 1 - \prod_p (1 - L^p(x))$$

5.6.5 Analyzing Training Errors

When the sensor tree wrongly labels a training example, the system analyzes this error and updates the confusion database if necessary. The confusion database, as described in section 5.6.2 keeps track of the possible interpretations of a sensor's output. This database is initially empty and new entries are added after observing the training errors.

We identify two types of errors. The first type is due to structural deficiency in the neural network. For example, there may exist instances of the feature $\mathcal{F}$ where none of the sensor sequences in the tree can detect. This type of error cannot be fixed by tuning the network parameters. The introduce of new interpretations for the existing sensors (hence new paths and potentially new sensors in the tree) on the other hand can potentially address such errors. Our error analysis procedure focuses on addressing this type of errors.

The second type of errors is due to inadequate parameters (e.g. the weights) in the neural network. This may either be caused by under-optimized parameters or structural deficiency in the neural net. We belief that when the network is structurally "adequate", even parameters far from the global optimum can perform well. Therefore we argue that by fixing the first type of errors, we also alleviate the second type of errors.

We handle the positive and negative examples differently. Positive examples are those in the

target class where feature $\mathcal{F}$ is expected to be present. Note that all positive examples (even those that are correctly labeled — but perhaps not for the right reason) participate in this procedure. The following steps are used to update the confusion database using the positive examples:

- For each positive example

    1. Traverse the sensor tree (beginning from the root) by following the correct interpretation given by the explanation.
    2. If the leave node is reached and the target object $\mathcal{T}^{\mathcal{F}}$ is detected, skip to the next example.
    3. If at node i, there is no outgoing edge that corresponds to the correct interpretation for S_i
        (a) Add an entry for S_i to the confusion database.
        (b) Stop traversing the tree and go to the next example.

It is possible that some sensor outputs do not have interpretations in the domain theory. We define an "unknown" object and use this as the interpretation for such sensor outputs.

For negative examples, we only process those that are wrongly labeled. In each of these examples, there must exist at least one sensor sequence that triggers the false detection. The following steps are used to update the confusion database using the negative examples:

- For each negative example

    – For each path with false activation, traverse the path beginning from the root node.

        1. At each node i, examine the list of expected interpretations for S_i.
        2. If the correct interpretation is not in the list, add an entry to the confusion database and skip to the next path.
        3. Otherwise, process the next node in the path.

This error analysis process ensures that any unexpected interpretations of a sensor (given its conditions) are recorded and will be added to any future trees that employ the same sensor under the same conditions. This is the source of information for *structural* refinement in the learning process.

5.6.6 Stopping Condition

It is straightforward to show that, for a given feature $\mathcal{F}$, with only a finite number of abstract objects and sensors in the domain theory, the space of possible sensor trees is finite. This means that we will eventually run out of new trees to build and the algorithm is forced to stop.

In practice, however, sensor trees that perform well can be constructed after very few iterations of the algorithm. Therefore, a simple threshold on the training error can be defined and the algorithm is stopped when K such trees have been constructed. K can be 1 if we do not intend to combine multiple trees for the same $\mathcal{F}$ in the final classifier.

5.6.7 A Refinement to the Algorithm

The basic learning algorithm requires that parameter optimization is performed for each constructed tree. This could result in waste of computational power on many "bad" sensor trees. Instead, we can perform a search through a series of candidate trees and only proceed to the parameter optimization step when a "structurally adequate" tree is found.

The followings are the two sources for alternative trees that do no require actually evaluating the optimized sensor tree:

- Step 5b in the tree construction algorithm (see section 5.6.2) does not specify any way to resolve ties. We can keep track of such nodes in the sensor tree, and use a simple breadth-first search to enumerate all alternative choices.
- Structural errors in the positive examples can be analyzed *without* first optimizing the parameters. This results in updates of the confusion database and the tree construction step can be repeated to generate new trees (as long as new updates are made in the confusion database).

The two options above can be used together to generate multiple candidate trees without going through parameter optimization. To decide whether a tree is "structurally adequate", we analyze the positive examples and count the number of structural errors (as determined by the explanation). A simple threshold (e.g. 90%) can be used to decide when a tree is structurally adequate.

5.7 Experiments

We evaluate our learning algorithm on the task of distinguishing difficult pairs of Chinese characters. We use the ETL9B database of offline handwritten Chinese characters. For each character there are 200 examples, each is a 63x64 binary image. As in previous chapters, we focus on binary classification tasks, each targeting a pair of similar (i.e. hard to distinguish) characters.

5.7.1 Domain Theory

For this algorithm, the domain theory needs to provide three components. First, for each object class, a set of abstract objects (with the associated appearance model) needs to be specified. We define the following three types of objects:

1. A stroke object, modeled as a line (or curve) segment with a fix width.
2. A "stroke segment" object, modeled as a (polygonal) patch of image that corresponds to pixels of a stroke. For a stroke that does not intersect any other strokes, the entire stroke itself creates a stroke segment object. For a stroke that is split into different parts due to intersecting with other strokes, each part creates a separate stroke segment.
3. A "stroke segment edge" object, modeled as a directed line (or curve) segment. For each stroke segment object, two edge objects are created (one for each side).

Figure 5.8 illustrates all 3 types of objects.

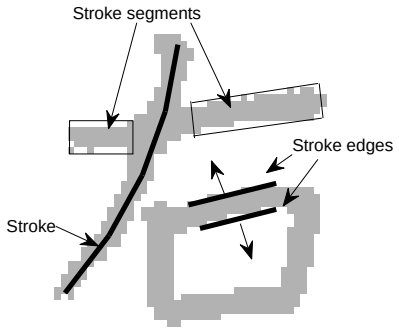


Figure 5.8: Examples object parts

Secondly, for each type of objects, one of more sensors need to be defined. We use the following sensors, each targets an object type defined above.

1. A line segment extractor based on Hough transforms. Sensor outputs are obtained by performing a Hough transform on an input image and parsing the image for line segments, each corresponds to a peak in the Hough transform.

2. A "blob" sensor for detecting stroke segments. The blob sensor depends on the conditional objects. For each conditional object that has a side constraint from the character model (see previous chapters), a line that divides the image plane into two halves is defined. The intersection of these half-planes then define a polygonal region from which an image patch is obtained. This patch becomes the sensor output. For example, in Fig. 5.8, given that the diagonal stroke has been detected, the stroke segment on its left can be detected by a (conditional) blob sensor.

3. An edge sensor for detecting edges. Sensor outputs are obtained by performing edge detection on the input image and extracting straight edge segments that are longer than a fixed threshold.

The last component needed is a set of consistency metrics associated with each object as well as each pair of objects. Note that not all pairs of objects need to have consistency metrics. We use the constraints from the character models as the source of consistency metrics, which include the followings:

- For strokes and edges, we define functions that measure the deviation from the ideal direction and location as defined in the character model.

- For pairs of strokes and/or edges, we define consistency metrics for their relative angles as well as relative positions.

- For pairs of strokes and/or edges where there exists a side constraint, we define consistency metrics that penalize violating configurations by the minimum distance needed to move each part into an acceptable configuration.

- For stroke segments, we define a consistency metric that measures the ratio between the expected number of dark pixels and number of detected dark pixels.

5.7.2 An Illustration

We illustrate how the algorithm performs on the pair of characters from Figure 5.1. The semantic feature that we try to detect is as defined in Figure 5.7.

We run the learning algorithm for 50 iterations and choose the tree that has the best training error rate. We use the proposed refinement to the algorithm such that in each iteration, a breadth-first search in the tree space is performed to find the smallest tree among 30 candidate trees. Figure 5.9 shows the very first sensor tree that is constructed. We also pre-analyze the positive examples and use a 90%-threshold to decide whether to proceed with parameter optimization or to repeat the breadth-first search. Therefore, although we only perform parameter optimization for 50 trees, many more trees were actually constructed.

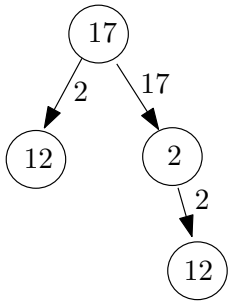


Figure 5.9: Example of a first tree

Figure 5.10 shows an example tree that is chosen at the end of the training process. The corresponding part labels are shown in Fig. 5.11.

We note an unexpected, but interesting surprise in the learned tree in Fig. 5.10, where one of the paths ends with an additional sensor for part no. 3, instead of stopping at the "target" part no. 12. A further examination of this shows that the additional sensing provides additional evidence for resolving potential uncertainty, which resulted in errors in some previously constructed trees. This illustrates the ability of the algorithm to enhance robustness through structural refinement.

We perform classification with two different versions of labeling, one with a fixed $\varphi = 0.5$ and the other with optimized φ for each sensor tree path. Table 5.1 compares the results we obtained with the sensor tree algorithm with those from the SVMs. The error rates are based on 5-fold cross-validation.

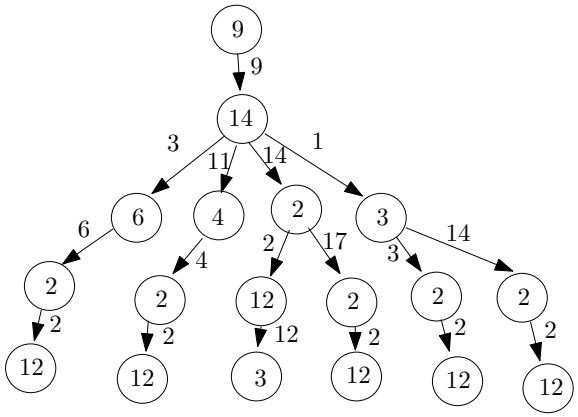


Figure 5.10: Example final tree

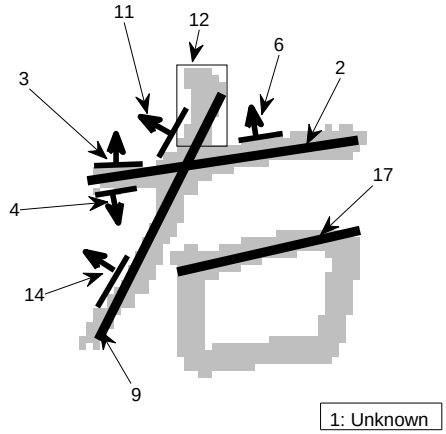


Figure 5.11: Object part labels for the example trees

5.7.3 Adjusted Training Error

Given the explanations for the training examples, we can actually evaluate training errors in two different ways. The first, "raw" training error is simply the error rate based on the labels. However, for each positive example, we know the exact path in the sensor tree that correctly detects the intended semantic feature. If the example is correctly labeled but based on a different path in the tree, we know that the sensor tree does not work for the right reason on that particular example. Using this information, we define a second, adjusted training error where an example is counted as correctly labeled only when it is triggered by the correct path in the sensor tree. Fig-

Learner	Error Rate (%)
SVM (raw image)	11.25
SVM (WDH features)	5.5
Simple Semantic Feature	5.25
Sensor tree ($\varphi = 0.5$)	2.75
Sensor tree (φ optimized)	2

Table 5.1: Classification Error Rate (5-fold cross-validation)

ures 5.12 and 5.13 show the two kinds of errors for our pair of characters, measured at each iteration of the algorithm on a newly constructed sensor tree. Plotted on the same figures are the test errors, which are not observable during training.

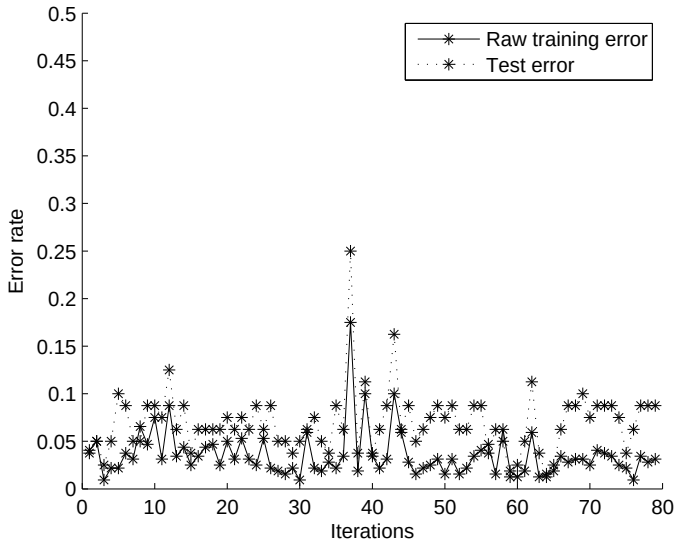


Figure 5.12: Raw training error

The raw training error shows noticeable overfitting especially during later iterations when larger sensor trees are used. On the other hand, the adjusted training error remains very similar to the actual test error. This shows that the adjusted training error is a strong indicator for the actual test error, and this is a significant piece of evidence that working for the right reason implies good generalization.

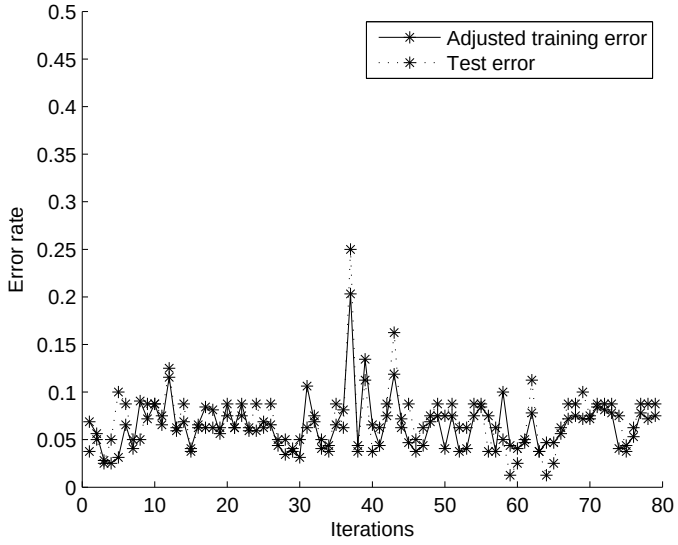


Figure 5.13: Adjusted training error

Chapter 6

A Unifying View and Comparison of the Three Approaches

The proposed approaches of feature and model construction can be viewed as different ways of incorporating prior knowledge into supervised learning. Although the final classifier training is discriminative, we observe both generative and discriminative elements in the prior domain knowledge itself as well as the in the feature training process. This has consequences in terms of the kind of domain theory required, the computational cost involved, the amount of prior information utilized by the learning system, and of course the final performance. Ultimately, we would like to answer questions such as "given the type of prior knowledge I have, which approach should I use?"

We use the shorthands PHANTOM, SF-SIMPLE and SF-COMPLEX to refer to the phantom examples approach, and the two feature construction approaches respectively. Table 6.1 provides a summary of the various aspects of learning involved.

	Prior Knowledge	Feature training	Classifier training
PHANTOM	Generative	Generative	Discriminative
SF-SIMPLE	Generative+Discriminative	Generative	Discriminative
SF-COMPLEX	Generative+Discriminative	Generative+Discriminative	Discriminative

Table 6.1: Generative and discriminative elements in the proposed approaches

Figure 6.1 shows a conceptual view of the three approaches in terms of the interaction between prior domain knowledge and the training examples.

6.1 Generative and Discriminative Prior Knowledge

Generative prior knowledge usually means information about the joint distribution $\Pr(X, Y)$, in particular, the distribution in the input X. A particularly natural form of generative knowledge is in terms of the input distribution given the object class $\Pr(X|Y)$. For Chinese characters this would mean given a specific character, we have information about how its instances are dis-

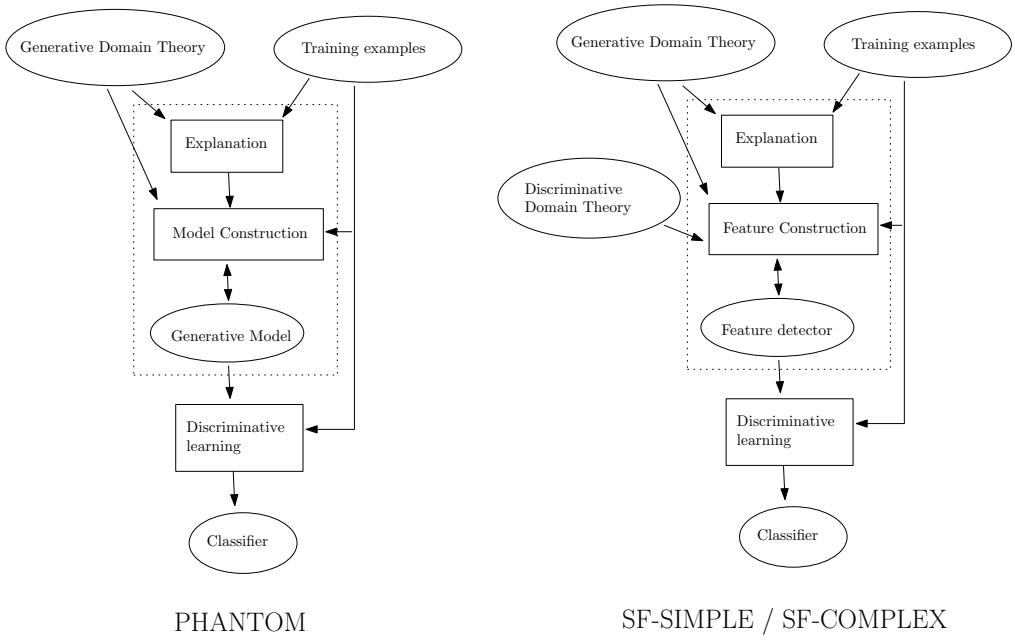


Figure 6.1: Conceptual view of the learning systems. The dotted box shows where prior domain knowledge and training examples interact.

tributed. Decomposition of objects into parts and different levels of abstraction provides further information in a generative sense. For example, from character to strokes, then strokes to pixels.

Discriminative prior knowledge is less common. It is information on $\Pr(Y|X)$, which has direct connection to the classification decision surface itself. In other words, given input X, discriminative knowledge provides information about the locations of the decision boundaries.

The decision boundaries, intuitively, must be located where the *differences* between class are. Although this boundary may be very complex in the raw input space (e.g. images), it may be much simpler at a more abstract level (e.g. in terms of character strokes). In our feature construction algorithms, we exploit this fact and perform a between-class comparison at the stroke level to locate potentially useful differences. Subsequent learning can then focus on only these informative regions.

6.2 Generative and Discriminative Feature Training

By "feature training", we mean how the training data is used to find the best-fitting model or feature transformations. In all approaches this involves an optimization process. The distinction between generative and discriminative training can be made clear by looking at the objective func-

tion used in the individual optimization process.

The "explanation" process, which reveals the hidden structure within each input (e.g. strokes) is common to all three approaches, and it utilizes mainly generative information. In PHANTOM, a generative model is fitted to this information. By definition, this is generative training.

In SF-SIMPLE, a robust detector is learned to find both the reference as well as the target feature. Although the target feature is known to contain discriminative information, the detector is not optimized such that the resulting feature transform is discriminative. Again, this is generative training.

In contrast, SF-COMPLEX trains the feature detector in a discriminative manner. Here, we note that the objective function used in the optimization of the sensor tree parameters actually penalizes label loss.

6.3 Elements in Domain Theory

All EBL approaches require that we have a domain theory. This is our source of information in addition to the training data.

Common to all approaches is the object model that we use to describe each class. In particular, we utilize prototype characters which contain stroke-level information including qualitative constraints on possible stroke configurations. The connection between strokes and pixels is provided by a stroke appearance model (e.g. straight lines or curves). This information is sufficient for the PHANTOM approach.

For the two feature construction approaches, the domain theory is required to provide additional information, namely,

1. A mechanism to perform between-class similarity/difference comparison. This is assumed to operate at a high-level abstraction (e.g. stroke level) and is not required to be accurate. This is our source of semantic features.

2. Sensors whose output can be interpreted into abstract object parts (e.g. strokes or parts of stroke).

SF-COMPLEX has a further requirement that each sensor-interpretation pair can be quantitatively evaluated, through one or more consistency metrics. The consistency metrics, however, are expected to be approximate and imperfect.

We observe that the more complex approaches require more information from the domain theory. While this means that more information can be utilized by these approaches, they are also more susceptible to any imperfections in the domain theory. This tradeoff is certainly expected and in practice we can make our decision based on the level of confidence we have in our domain theory.

This said, our empirical results indicate that all the approaches are generally robust with respect to imperfection in the domain theory.

6.4 Computational Complexity

We note that for all our approaches, the computational effort is mainly spent during the training phase. Once trained, the classifier can evaluate new inputs efficiently, with at most polynomial growth in the computational time with respect to the solution size.

In training, most computational effort is spent during the explanation and the feature training phase. The computational complexity of the final discriminative learner is relatively insignificant since for SVM training the worst case time complexity is still polynomial ($\mathcal{O}(m^3)$ where m is the size of the training set). The exception is PHANTOM since in principle, we can add unlimited number of phantom examples to the training set but in practice, the storage complexity ($\mathcal{O}(m^2)$) may quickly become prohibitive.

The explanation process largely depends on the structure of our character prototypes, where for a general graph it is intractable. However, this might not be a problem if every stroke-level interaction is limited to a small set, which is usually the case in practice.

For SF-COMPLEX, the feature training (sensor tree optimization) may involve non-convex optimization and can be computationally more costly than the other approaches. Our empirical observation is that the better the quality of the domain theory, the faster the optimization.

Chapter 7

Related Works

7.1 Motivation from Some Previous Works

This research was partly motivated by previous works by Sun [Sun, 2005]. In particular, the phantom examples approach can be considered an extension and generalization of the work in [Sun, 2005]. Also related is work in [Brodie and DeJong, 2001] that used phantom examples for control learning.

The proposed feature construction algorithms are related to the feature kernel functions proposed in [Sun and DeJong, 2005] in the sense that both bias the discriminative learner to concentrate on informative regions in the input image. However, instead of using weights on absolute pixel coordinates through the kernels, our approach uses a more adaptive detection method, which can be viewed as a form of dynamic image registration. For an example, consider the pair of characters in 7.1.

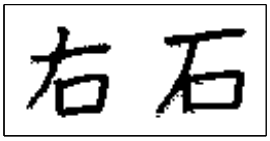


Figure 7.1: Example pair of similar characters

The top-left corner of the input image is the most informative for this pair. However, due to the large variability among instances of the same character, an absolute coordinate approach needs to define a sufficiently large region in order to cover most cases. This may result in loss of sensitivity to the target feature. An adaptive approach would ameliorate this problem. Figure 7.2 illustrates this difference.

The adaptive detection, however, cannot be easily translated into an SVM kernel, hence our proposed feature construction algorithms. We note that the jittered SV method [Decoste and

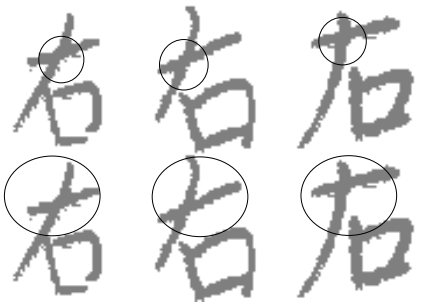


Figure 7.2: Top row: adaptive concentration. Bottom row: absolute concentration.

Schölkopf, 2002] has some flavor of dynamic image registration, however, the jittering is done to the *entire* image and therefore does not take advantage of any particularly informative parts of the input.

7.2 Artificial Training Examples

Approaches that use invariant transformations to augment the training data include [Pomerleau, 1991; Baird, 1992; Poggio and Vetter, 1992; Burges and Schölkopf, 1996; Roth et al., 2000; Decoste and Schölkopf, 2002]. Our phantom example approach can be seen as a form of generalization of these approaches where an underlying generative model is assumed. With this view, we admit the incorporation of richer domain knowledge through the construction of task-specific models.

In the context of object recognition and computer vision, many approaches that employ synthetic graphics data have been explored. In [Sapp et al., 2008], synthetic images of objects in different background/foreground combination are generated with a trained generative model. [Michels et al., 2005] use 3D models of outdoor scenes to generate images for autonomous driving. In [Heisele et al., 2001] and [Everingham and Zisserman, 2005], textured 3D head models are used to generate faces in arbitrary poses and illumination. [Miyao and Maruyama, 2006] use online handwriting data to learn a generative model for Chinese characters.

Most of the approaches mentioned above rely on a generative model that is handcrafted for the specific tasks. Our model construction algorithm provides a way to partially automate this process by employing available domain knowledge as a model generator and use the training examples to guide the construction of task-specific models.

7.3 Feature Generation and Selection

Most works on feature construction can be seen as a combined feature generation-selection approach. The feature generation part either explicitly or implicitly defines a space of possible features. The most useful features within this space are then selected using some data-dependent criteria. The feature selection methods include filters [Almuallim and Ditterich, 1991; Kira and Rendell, 1992], wrappers [Kohavi and John, 1997], as well as embedded methods [Guyon et al., 2006]. Central to the feature selection approaches is the notion of relevance [Blum and Langley, 1997; Kohavi and John, 1997], which is the basis for deciding the usefulness of any given feature.

Arguably the simplest form of this generate-selection approach is feature subset selection from a predefined list of all potential features. A simple extension to this is to consider linear combinations of the existing features. The standard principal components analysis (PCA) and linear discriminant analysis (LDA) are examples of this. To address nonlinearity, kernel-based extensions to these algorithms have also been proposed (e.g. kernel PCA [Schölkopf et al., 1996]).

An observation of these feature selection approaches is that they are generally oblivious to any task or domain specific information, other than what have been captured by the given features. In contrast, our semantic features are mainly derived from a domain theory in a task specific manner through the explanation-based interaction with the training examples. An "analytical" component has been added to the otherwise empirically guided process.

Domain-aware features have been used in the context of constructive induction and inductive logic programming, where features are logical formulas. Using logical operators, an expressive space of potential features can be generated from basic predicates. Works in this regard include [Markovitch and Rosenstein, 2002; Utgoff, 2001; Flach and Lavrac, 2000; Hu and Kibler, 1996; Fawcett and Utgoff, 1992]. We note that many of these approaches intend to *invent* new features in a bottom-up, data-driven manner. Our semantic features, on the other hand, are derived from the domain theory in a top-down manner and instead of finding new semantic features, training data is used to resolve any uncertainties between the semantic features and the observed features.

Our semantic features have resemblance to the "types" of features defined in [Cumby and Roth, 2002, 2003], which can be seen as high-level generalizations of lower-level ground features. In their works, formal syntax and semantics are defined for these features, which allows analysis in terms of representation expressiveness and computational complexity. We do not define such formal representations for our features, but concentrate on more interaction between data and the domain theory in a complex domain.

7.4 Features in Object Recognition

Features play an important role in object recognition. One particularly popular direction of research is in designing *invariant* features. Features that are robust under transformations such as scaling, rotation, illumination changes or 3D projection have great potential of being able to generalize to unseen configuration of objects. Many such features have been proposed, examples include those based on corner detectors [Schmid and Mohr, 1997; Zhang et al., 1995], SIFT-based features [Lowe, 1999; Lazebnik et al., 2004], Haar wavelets [Mohan et al., 2001] or Gabor filters [Serre et al., 2005].

A particularly successful approach to feature construction is via the use of a hierarchy, where intermediate, or hidden features are used. Some of these approaches are inspired by the hierarchical nature of the visual cortex [Serre et al., 2005; Mutch and Lowe, 2006]. Other hierarchical approaches include convolutional networks [Lecun et al., 1998; Hua] and hierarchical SVM classifiers [Heisele et al., 2001]. In many approaches, intermediate features are associated with object parts or components such that their spatial relationships can be exploited in recognition [Agarwal and Roth, 2002; Ullman et al., 2002; Heisele et al., 2001; Weber et al., 2000]. More recently, unsupervised training of individual feature layer has been shown to be promising [Bengio et al., 2007; Ranzato et al., 2007].

Many, if not most of these features are intended to be generic and insensitive to the underlying task objects or the meaning of individual object parts. These features generally work well when the available training set is large but can perform poorly otherwise. What our approaches bring to this is the incorporation of task-specific knowledge about the underlying objects into the feature construction process. In particular, our semantic features specify relationships among object parts in terms of their well-formedness, and our feature construction process involves interpretation of lower-level sensor outputs in terms of these object parts.

7.5 Handwritten Chinese Character Recognition

There has been a huge body of literature on handwritten Chinese character recognition. Two major approaches are the structural and the feature-based approach [Suen et al., 2003]. Structural approaches, e.g, [Liu et al., 2001] involve the extraction of possible strokes in a given image and the matching of the extracted strokes to existing templates. The major weakness of this approach comes from noise in the stroke-extraction process, which often result in poor matchings, especially

for similar characters. The feature-based approaches, e.g, [Tang et al., 1998] rely on extracting a number of features (e.g. directional features, stroke density, etc) and build a classifier based on the statistics of these features. These approaches are usually more robust but require a large number of training examples, which is rarely available for handwritten Chinese characters. See [Suen et al., 2003; Arica and Yarman-Vural, 2001] and references therein for more works in this regard.

Our approach to Chinese character recognition is to decompose the problem into smaller tasks (e.g. binary classification between difficult pairs). One such decomposition is via a two-phase approach, where a coarse classifier is used as the first phase. A coarse classifier can typically detect the top k most likely candidates for each input with very high accuracy for some small k. The second phase involves more fine-grained classifiers that are specifically trained to distinguish between mutually confusing characters. It would be too tedious to manually design features for each of these subtasks and our automated feature construction algorithms can be of significant help.

Chapter 8

Conclusions and Future Works

We conclude with a summary of the contributions of this work and a brief discussion of potential future works.

8.1 Contributions

The main contribution of this work is in taking a few first steps in addressing the critical but under-explored area of automated model and feature construction, and the incorporation of prior domain knowledge into this process. In particular, we proposed three approaches for incorporating both generative and discriminative knowledge into discriminative learning:

- *An automated model construction algorithm for phantom examples.* Available domain knowledge is used to construct a space of task-specific alternative models for describing the training examples in terms of their underlying, hidden structures. Once learned, the model is used to generate phantom examples that enhance subsequent discriminative learning.

- *Simple, discriminative semantic features.* Available domain knowledge is used to derive potentially useful semantic features by exploiting high-level similarities and differences between classes of objects. Efficient and robust detectors for these features are learned through explanation-based interaction between the training data and the domain knowledge.

- *Complex discriminative semantic features.* An extension to the simple semantic feature is proposed where more expressive sensor trees are used as the feature detectors. Our algorithm uses various interpretations of sensor outputs with respect to the underlying object parts in conjunction with the explanations of the training examples to ensure that the constructed sensor trees actually detect the intended semantic features.

The proposed algorithms are all motivated by the idea of "working for the right reason" and "well-formed concepts", which we view as key guiding principles for the ability to generalize in

supervised learning. In particular, we contributed a novel view of generalization by considering the interaction between prior knowledge and training data using information theory and PAC-Bayesian analysis.

In the domain of offline handwritten Chinese character recognition, we made contributions in addressing a particularly challenging problem of distinguishing similar characters. All our proposed algorithms are shown to be effective in this regard and achieve performance level on par or better than the state-of-the-art solutions.

8.2 Future Works

8.2.1 Formal Representation of the Semantic Features

We did not insist on any particular form of representation for our domain theory or the semantic features. A well-defined syntax and semantics, however, will allow further analysis in terms of adequacy of a domain theory and its impact on the derived features and their ability to generalize. This also enables analysis on the tradeoff between expressiveness of representation and computational complexity.

8.2.2 Partial and Incremental Explanation

Our proposed approaches assume a separate explanation process independent of the actual feature construction itself. This also implies that we need to fully explain each training example, which can be computationally costly. A combined or embedded approach, with the ability to perform partial and incremental explanation in an on-demand fashion may significantly reduce this cost.

8.2.3 Document-level Recognition

We focused on fine-grained features in our classification tasks. Scaling up to a document-level recognition task is a potential next step, where coarser-grained features can be employed. This also opens the doors for incorporating other sources of information such as a language model.

8.2.4 Other Problem Domains

Many closely related domains can be explored with relatively little modification of our domain theory. These include recognition of other Asian scripts, as well as line drawings classification.

Other, more general object recognition tasks may require more substantial change in the domain theory, but will provide further insight into the generality of the proposed approaches across different application domains.

References

Shivani Agarwal and Dan Roth. Learning a sparse representation for object detection. In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part IV*, pages 113–130, London, UK, 2002. Springer-Verlag. ISBN 3-540-43748-7.

H Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, (19):716–723, 1974.

M Akaike and N Merhav. Relations between entropy and error probability. *IEEE Transactions on Information Theory*, 40(1):259–266, 1994.

H Almuallim and T G Ditterich. Learning with many irrelevant features. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 547–552. AAAI Press/The MIT Press, 1991.

N. Arica and F. T. Yarman-Vural. An overview of character recognition focused on off-line handwriting. *Systems, Man and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 31(2):216–233, 2001. doi: http://dx.doi.org/10.1109/5326.941845. URL `http://dx.doi.org/10.1109/5326.941845`.

H S Baird. Document image defect models. In *Structured Document Analysis*, pages 1–16. Springer-Verlag, 1992.

P Bartlett and S Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, (3), 2002.

R Battiti. Using the mutual information for selecting features in supervised neural net learning. *IEEE Transactions on Neural Networks*, (5):537–550, 1994.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 153–160. MIT Press, Cambridge, MA, 2007.

AL Blum and P Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, (97):245–271, 1997.

Mark Brodie and Gerald DeJong. Iterated phantom induction: A knowledge-based approach to learning control. *Machine Learning*, 45(1):45–76, Oct 2001. doi: 10.1023/A:1010976022071. URL `http://dx.doi.org/10.1023/A:1010976022071`.

Christopher J. C. Burges and Bernhard Schölkopf. Improving the accuracy and speed of support vector machines. In *NIPS*, pages 375–381, 1996.

C. H. Chen. *Handbook Of Pattern Recognition And Computer Vision*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2005. ISBN 9812561056.

C. Cumby and D. Roth. Learning with feature description logics. In *Proc. of the International Conference Inductive Logic Programming*, pages 32–47, 2002. URL `http://l2r.cs.uiuc.edu/ danr/Papers/ilp02.pdf`.

C. Cumby and D. Roth. Feature extraction languages for propositionalized relational learning. In *IJCAI Workshop on Learning Statistical Models from Relational Data*, 2003.

Dennis Decoste and Bernhard Schölkopf. Training invariant support vector machines. *Mach. Learn.*, 46(1-3):161–190, 2002. ISSN 0885-6125.

Gerald DeJong. Toward robust real-world inference: A new perspective on explanation-based learning. In *In Proc. ECML/PKDD. Vol. 4212 of LNCS*, pages 102–113. Springer Verlag, 2006.

Gerald DeJong and Raymond Mooney. Explanation-based learning: An alternative view. In *Machine Learning*, pages 145–176, 1986.

Luc Devroye, Laszlo Györfi, and Gabor Lugosi. *A Probabilistic Theory of Pattern Recognition*. Springer, 1996. ISBN 9780387946184.

Mark Everingham and Andrew Zisserman. Identifying individuals in video by combining "generative" and discriminative head models. In *ICCV '05: Proceedings of the Tenth IEEE International Conference on Computer Vision*, pages 1103–1110, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2334-X-02. doi: http://dx.doi.org/10.1109/ICCV.2005.116.

Tom E. Fawcett and Paul E. Utgoff. Automatic feature generation for problem solving systems. Technical report, Amherst, MA, USA, 1992.

P Felzenszwalb and D Huttenlocher. Efficient matching of pictorial structures. In *Proc. IEEE Computer Vision and Pattern Recognition Conf., 2000*, pages 66–73, 2000.

Peter A. Flach and Nada Lavrac. The role of feature construction in inductive rule learning. Technical report, Bristol, UK, UK, 2000.

David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002. ISBN 0130851981.

E Gabrilovich and S Markovitch. Feature generation for text categorization using world knowledge. In *In IJCAI 05*, pages 1048–1053, 2005.

I.M. Guyon, S.R. Gunn, M. Nikravesh, and L. Zadeh. *Feature Extraction, Foundations and Applications*. Springer, 2006.

B. Heisele, T. Serre, M. Pontil, T. Vetter, and T. Poggio. Categorization by learning and combining object parts, 2001. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.21.9269`.

Ralf Herbrich and Thore Graepel. A pac-bayesian margin bound for linear classifiers: Why svms work. In *In Advances in Neural Information Processing Systems 13*, pages 224–230. MIT Press, 2001.

Y Hu and D Kibler. Generation of attributes for learning algorithms. In *Proc. 13th International Conference on Machine Learning*, pages 806–811. Morgan Kaufmann, 1996.

S Impedovo, P Wang, and H Bunke. *Automatic Bankcheck Processing*. World Scientific, 1997.

Tommi S. Jaakkola and David Haussler. Exploiting generative models in discriminative classifiers. In *Proceedings of the 1998 conference on Advances in neural information processing systems II*, pages 487–493. MIT Press, 1999. ISBN 0-262-11245-0.

F. Kimura. Improvement of handwritten japanese character recognition using weighted direction code histogram. *Pattern Recognition*, 30(8), 1997.

Kenji Kira and Larry A. Rendell. A practical approach to feature selection. In *ML*, pages 249–256, 1992.

Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artif. Intell.*, 97(1-2): 273–324, 1997. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/S0004-3702(97)00043-X. URL `http://dx.doi.org/10.1016/S0004-3702(97)00043-X`.

Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Semi-local affine parts for object recognition. In *In BMVC*, pages 959–968, 2004.

Yann Lecun, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

Ruei-Sung Lin, David A. Ross, Jongwoo Lim, and Ming-Hsuan Yang. Adaptive discriminative generative model and its applications. In *Advances in Neural Information Processing Systems 17*, pages 801–808. MIT Press, 2005.

C-L Liu, I-J Kim, and J H Kim. Model-based stroke extraction and matching for handwritten chinese character recognition. *Pattern Recognition*, (34):23392352, 2001.

D. G. Lowe. Object recognition from local scale-invariant features. volume 2, pages 1150–1157 vol.2, 1999. doi: http://dx.doi.org/10.1109/ICCV.1999.790410. URL `http://dx.doi.org/10.1109/ICCV.1999.790410`.

S Markovitch and D Rosenstein. Feature generation using general constructor functions. In *Machine Learning*, pages 59–98. The MIT Press, 2002.

David A. Mcallester. Pac-bayesian stochastic model selection. In *Machine Learning*, page 2003, 2003.

David A. Mcallester. Pac-bayesian model averaging. In *In Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, pages 164–170. ACM Press, 1999.

Jeff Michels, Ashutosh Saxena, and Andrew Y. Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 593–600, New York, NY, USA, 2005. ACM. ISBN 1-59593-180-5. doi: http://doi.acm.org/10.1145/1102351.1102426.

T Mitchell, R Keller, and S Kedar-Cabelli. Explanation-based generalization: A unifying view. *Mach. Learn*, (1):47–80, 1986.

Tom M. Mitchell. The need for biases in learning generalizations. Technical report, 1980.

Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

Hidetoshi Miyao and Minoru Maruyama. Virtual example synthesis based on pca for off-line handwritten character recognition. In *Document Analysis Systems*, pages 96–105, 2006.

Anuj Mohan, Constantine Papageorgiou, and Tomaso Poggio. Example-based object detection in images by components. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23: 349–361, 2001.

R Moratz, J Renz, and D Wolter. Qualitative spatial reasoning about line segments. In *ECAI 2000. Proceedings of the 14th European Conference on Artifical Intelligence*, pages 234–238. IOS Press, 2000.

Jim Mutch and David G. Lowe. Multiclass object recognition with sparse, localized features. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 11–18, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2597-0. doi: http://dx.doi.org/10.1109/CVPR.2006.200.

A. Ng and M. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes, 2002. URL `citeseer.ist.psu.edu/542917.html`.

Tomaso Poggio and Thomas Vetter. Recognition and structure from one 2d model view: Observations on prototypes, object classes and symmetries. *Laboratory, Massachusetts Institute of Technology*, 1347, 1992.

Dean A. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3:97, 1991.

Lionel Prevost, Loïc Oudot, Alvaro Moises, Christian Michel-Sendis, and Maurice Milgram. Hybrid generative/discriminative classifier for unconstrained character recognition. *Pattern Recogn. Lett.*, 26(12):1840–1848, 2005. ISSN 0167-8655. doi: http://dx.doi.org/10.1016/j.patrec.2005.03.005.

A. Quattoni, S. Wang, L. p Morency, M. Collins, T. Darrell, and Mit Csail. Hidden-state conditional random fields. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007.

Rajat Raina, Yirong Shen, Andrew Y. Ng, and Andrew McCallum. Classification with hybrid generative/discriminative models. In *Advances in Neural Information Processing Systems 16*. MIT Press, 2004.

Marc' A. Ranzato, Christopher Poultney, Sumit Chopra, and Yann Lecun. Efficient learning of sparse representations with an energy-based model. In *NIPS*, 2006. URL `http://nips.cc/Conferences/2006/Program/event.php?ID=425`.

Marc'Aurelio Ranzato, Fu-Jie Huang, Y-Lan Boureau, and Yann LeCun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *Proc. Computer Vision and Pattern Recognition Conference (CVPR'07)*. IEEE Press, 2007.

Matthew Richardson and Pedro Domingos. Markov logic networks. In *Machine Learning*, page 2006, 2006.

J Rissanen. Modeling by shortest data description. *Automatica*, (14):465–471, 1978.

Dan Roth, Ming hsuan Yang, and Narendra Ahuja. A snow-based face detector. In *Advances in Neural Information Processing Systems 12*, pages 855–861. MIT Press, 2000.

William Rucklidge. *Efficient Visual Recognition Using the Hausdorff Distance*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996. ISBN 3540619933.

Benjamin Sapp, Ashutosh Saxena, and Andrew Y. Ng. A fast data collection and augmentation procedure for object recognition. In *AAAI*, pages 1402–1408, 2008.

Cordelia Schmid and Roger Mohr. Local grayvalue invariants for image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):530–535, 1997. ISSN 0162-8828. doi: http://dx.doi.org/10.1109/34.589215. URL `http://dx.doi.org/10.1109/34.589215`.

Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem, 1996.

G Schwarz. Estimating the dimension of a model. *Annals of Statistics*, (6):461–464, 1978.

Thomas Serre, Lior Wolf, and Tomaso Poggio. Object recognition with features inspired by visual cortex. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2*, pages 994–1000, Washington, DC, USA, 2005. IEEE Computer Society. doi: 10.1109/CVPR.2005.254. URL `http://dx.doi.org/10.1109/CVPR.2005.254`.

Daming Shi, Robert I. Damper, and Steve R. Gunn. Offline handwritten chinese character recognition by radical decomposition. *ACM Transactions on Asian Language Information Processing (TALIP)*, 2(1):27–48, 2003. ISSN 1530-0226. doi: http://doi.acm.org/10.1145/964161.964163.

Patrice Y. Simard, Dave Steinkraus, and John C. Platt. Best practice for convolutional neural networks applied to visual document analysis. In *In International Conference on Document Analysis and Recogntion (ICDAR), IEEE Computer Society, Los Alamitos*, pages 958–962, 2003.

Sargur N. Srihari and Edward J. Kuebert. Integration of hand-written address interpretation technology into the united states postal service remote computer reader system. In *ICDAR '97: Proceedings of the 4th International Conference on Document Analysis and Recognition*, pages 892–896, Washington, DC, USA, 1997. IEEE Computer Society. ISBN 0-8186-7898-4.

Ching Y. Suen, Shunji Mori, Soo H. Kim, and Cheung H. Leung. Analysis and recognition of asian scripts - the state of the art. In *ICDAR '03: Proceedings of the Seventh International Conference on Document Analysis and Recognition*, page 866, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1960-1.

Q Sun and G DeJong. Feature kernel functions: Improving svms using high-level knowledge. *CVPR*, (2):177–183, 2005.

Qiang Sun. *Explanation-Based Approach to Incorporating Domain Knowledge into Support Vector Machine: Theory and Applications*. PhD thesis, University of Illinois, Urbana-Champaign, 2005.

Yuan Y. Tang, Lo-Ting Tu, Jiming Liu, Seong-Whan Lee, Win-Win Lin, and Ing-Shyh Shyu. Offline recognition of chinese handwriting by multifeature and multilevel classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):556–561, 1998. ISSN 0162-8828. doi: http://doi.ieeecomputersociety.org/10.1109/34.682186.

N Tishby, F C Pereira, and W Bialek. The information bottleneck method. In *In Proc. 37th Annual Allerton Conference on Communication, Control and Computing*, 1999.

Simon Tong and Daphne Koller. Restricted bayes optimal classifiers. In *Proc. of AAAI-00*, pages 658–664. AAAI Press / The MIT Press, 2000. ISBN 0-262-51112-6.

K Torkkola. Feature extraction by non-parametric mutual information maximization. *Journal of Machine Learning Research*, (3):1415–1438, 2003.

Shimon Ullman, Michel Vidal-Naquet, and Erez Sali. Visual features of intermediate complexity and their use in classification. *Nat Neurosci*, 5(7):682–687, July 2002. doi: 10.1038/nn870. URL `http://dx.doi.org/10.1038/nn870`.

Paul E. Utgoff. Feature construction for game playing. pages 131–152, 2001.

Markus Weber, Max Welling, and Pietro Perona. Unsupervised learning of models for recognition. In *ECCV '00: Proceedings of the 6th European Conference on Computer Vision-Part I*, pages 18–32, London, UK, 2000. Springer-Verlag. ISBN 3-540-67685-6.

David H. Wolpert. The supervised learning no-free-lunch theorems. In *In Proc. 6th Online World Conference on Soft Computing in Industrial Applications*, pages 25–42, 2001.

David H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8:1341–1390, 1996.

Zhengyou Zhang, Rachid Deriche, Olivier D. Faugeras, and Quang T. Luong. A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry. *Artificial Intelligence*, 78(1-2):87–119, 1995. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.52.9733`.

Curriculum Vitæ

Shiau Hong Lim

Department of Computer Science, 201 N. Goodwin Ave., 6616 Siebel Center

University of Illinois, Urbana, IL 61801, USA

E-mail: shonglim@gmail.com

RESEARCH INTERESTS

Machine learning, explanation-based learning, reinforcement learning, pattern recognition, computer vision and artificial intelligence.

EDUCATION

Ph.D. in Computer Science, University of Illinois at Urbana-Champaign, 2009

Thesis: *Explanation-Based Feature Construction*

Advisor: Gerald DeJong

M.C.S. (with Distinction), University of Malaya, 2001

Thesis: *Traffic Engineering Enhancement to OSPF for IP QoS with Diffserv and MPLS*

B.C.S. (First Class Honors), University of Malaya, 2000

Thesis: *Fuzzy Logic in ATM: Policer and Queue Control*

PROFESSIONAL POSITIONS

Program Committee IJCAI-2007 Workshop on Analytics for Noisy Unstructured Text Data

Research Assistant

- Dept. of Psychology, University of Illinois, (2009)

  Worked with Professor Regenwetter on research in quantitative psychology.

- Dept. of Computer Science, University of Illinois, (2005-2008)

  Worked with Professor DeJong on research in Explanation-Based Learning.

- Faculty of Computer Science and IT, University of Malaya, (2000-2002)

  Worked with Prof. Yaacob, Dr. Ling and Dr. Phang on research in computer networks.

Teaching Assistant

- Dept. of Computer Science, University of Illinois, (2002-2005, 2007-2008)
  Conducted lab sections for Introduction to Computing (CS 101 and CS 105).
  TA for CS 440 Artificial Intelligence.

AWARDS AND HONORARIES

- Excellent Teaching Assistant Award, Fall 2004.
- Excellent Teaching Assistant Award, Fall 2003.
- Excellent IT Student Award 2000 by Malaysian National Computer Confederation (MNCC).
- Anugerah Bestari Celcom for outstanding thesis in the field of telecommunications, computer science, information technology and engineering, 2000.

PRESENTATIONS

- Paper presentation, "Towards Finite-Sample Convergence of Direct Reinforcement Learning," at the European Conference on Machine Learning, Porto, Portugal, Oct 3-7, 2005.
- Paper presentation, "Explanation-Based Feature Construction" at IJCAI-07, India, 2007.

PUBLICATIONS

Journal Articles

1. Lim, S.H., Wang, L., DeJong, G. Integrating Prior Domain Knowledge into Discriminative Learning using Automatic Model Construction and Phantom Examples To appear in *Pattern Recognition*, 2009.
2. Sun, Q., Wang, L., Lim, S.H., DeJong, G. Robustness through prior knowledge: using explanation-based learning to distinguish handwritten Chinese characters *International Journal on Document Analysis and Recognition*, Vol 10, No. 3-4, pp. 175-186, Dec 2007.
3. Lim, S.H., Yaacob, M., Phang, K.K., Ling, T.C. Traffic engineering enhancement to QoS-OSPF in DiffServ and MPLS networks. *IEE Proc. on Communications*, Vol 151, No. 1, Feb 2004.

Refereed Conference Papers

4. Lim, S.H., Wang, L., DeJong, G. Integrating Prior Domain Knowledge into Discriminative Learning Using Phantom Examples. In *Proc. 11th International Conference on Frontiers in Handwriting Recognition*, 2008.

5. Lim, S.H., Wang, L., DeJong, G. Explanation-Based Feature Construction. In *Proceedings IJCAI-07*, 2007.

6. Lim, S.H., DeJong, G. Towards Finite-Sample Convergence of Direct Reinforcement Learning. In *Proceedings ECML 2005*, 2005.

7. Phang, K.K., Lim, S.H., Yaacob, M., Ling, T.C. Design of Fuzzy Usage Parameter Controller for Diffserv and MPLS. In *Proc. IEA/AIE 2002, Lecture Notes in Computer Science*, Vol. 2358, pp.470-481, 2002.

8. Lim, S.H., Yaacob, M., Phang, K.K., Ling, T.C. Fuzzy VBR Policer for ATM Networks. In *Proc. MMU International Symposium on ICT 2000*, 2000.